УДК 004.056.5 **Информационные технологии**

Вешкин Игорь Ильич, *студент 4 курса,*
факультета математики и информационных технологий
ФГБОУ ВО «НИ МГУ им. Н.П. Огарёва», г. Саранск

ЗАЩИТА ИНФОРМАЦИИ С ИСПОЛЬЗОВАНИЕМ JPEG-ИЗОБРАЖЕНИЙ

Аннотация: Предметом исследования является актуальный метод защиты информации с использованием изображений формата jpeg. В статье рассмотрен общий алгоритм работы записи и извлечения данных на примере рабочего программного кода языка программирования Python.

Ключевые слова: Python, jpeg-изображения, целевой файл, изображение-контейнер.

Abstract: The subject of the research is an relevant method of information protection using jpeg images. The article describes a general algorithm for recording and extracting data using the example of the working programming code written in Python programming language.

Keywords: Python, jpeg-picture, target file, container-picture.

Введение. Защита информации является важной частью современного мира. С переходом мира в новую цифровую эпоху данный вопрос стоит наиболее остро. Современная информация, в большинстве случаев, не имеет материального содержания (оболочки), а представляется в виде цифровых данных, вследствие чего способы кражи и получения доступа к ним совершенно изменились и требуют свежий индивидуальный подход.

В рамках данной статьи представлен довольно нестандартный метод защиты информации, в основе которого используются особенности формата

JPEG, а также его реализация с использованием языка программирования Python. Разработанный автором метод имеет много общего с принципами стеганография, науки изучающей и разрабатывающий методы секретной передачи данных, поскольку данный способ использует изображения в виде переносчика сторонних данных. При этом jpg-файл сохраняет все свои свойства, как объект данного расширения, и никоим образом не будет сигнализировать о наличии сторонней информации за исключением лишь объема. Отсюда следует главный недостаток метода – объем передаваемой информации не должен превышать объем исходного изображения, поскольку, например, изображение низкого разрешения и качества, занимающее 70 Мб в памяти компьютера, будет вызывать вопросы и подозрение.

Общие принципы функционирования метода. Возьмем совершено стандартное изображение с расширением jpg. Любой файл в вычислительной технике представим в виде последовательности байт, записанных по определенному правилу. Его еще называют байт-кодом. Существует ряд методов, используемых для просмотра байт-кода и его анализа, например, использование специализированного программного обеспечения (hex-редакторы). На рынке присутствуют свободные по распространению и использованию решения как в виде отдельных автономных приложений, так и в виде веб-приложений (сайтов), находящихся в сети Интернет.

Как можно видеть на рисунках 1 и 2, изображения формата JPEG, обладают двумя характерными маркерами: «FF D8» в начале и «FF D9» на конце. Эти маркеры являются, своего рода, константными значениями, которые обозначают начало и конец jpeg-файла [1].

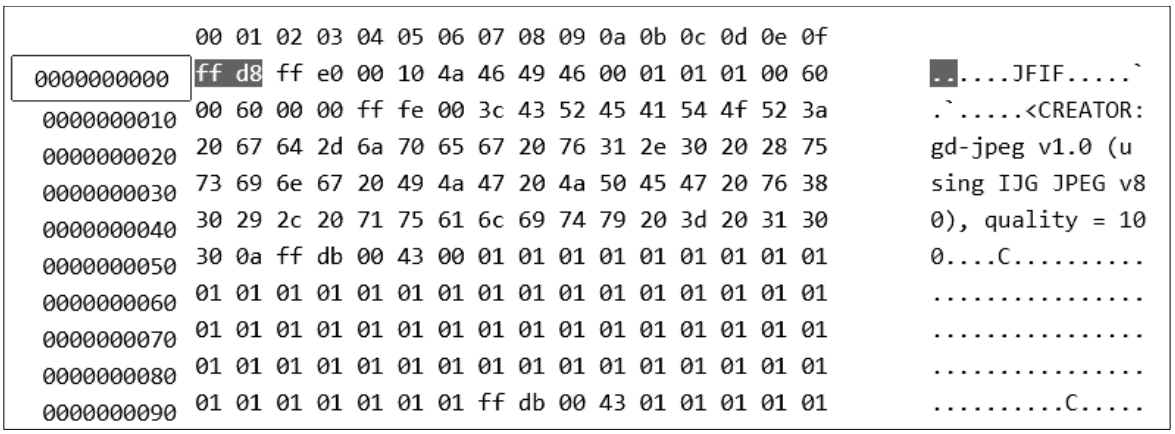


Рисунок 1 – Маркер начала последовательности байт

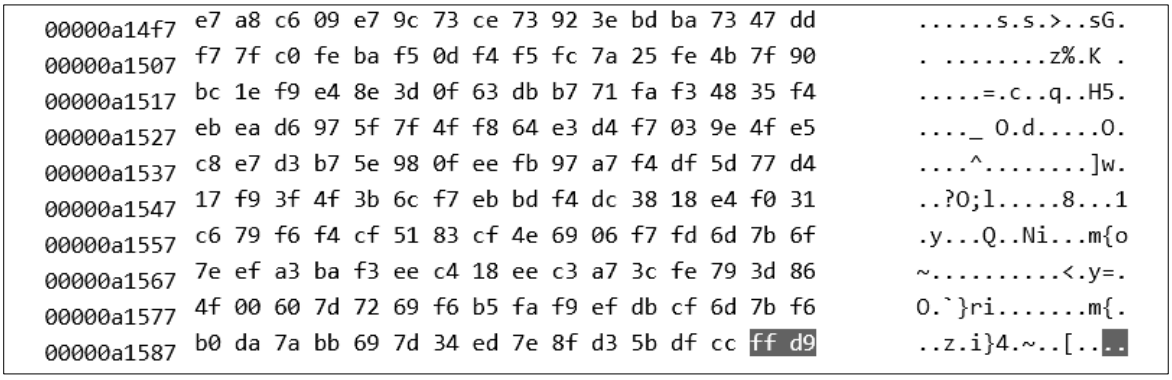


Рисунок 2 – Маркер конца последовательности байт

Внесение тех или иных изменений в байт-код, находящийся между маркерами, без должного понимания работы данного расширения, может привести к деформации тех или иных характеристик изображения, вследствие этого изменение внутренней части файла рассматриваться не будет. В рамках реализации разработанного метода защиты информации необходимо четко понимать: весь байт-код, записанный после «FF D9» маркера, обозначающий конец jpg-файла, будет игнорироваться. Новое изображение ничем не будет отличаться по своим внешним признакам от исходного, т. к. внутренняя часть не была изменена, но последовательность, дописанная после, будет храниться в файле на постоянной основе (рисунок 3). И, если новое изображение не на много превосходит по объему памяти исходника, то вероятность того, что дописанный код обнаружат достаточно низкая. И даже если это произойдет, без знания формата записанного в изображение файла, получить доступ к данным не представляется возможным, поскольку последовательность требуется считать, а

потом записать в файл правильного формата.

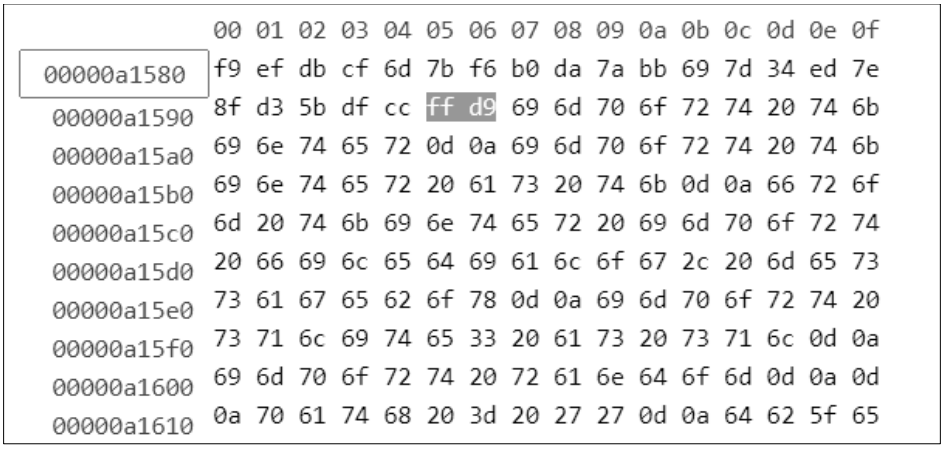


Рисунок 3 – Представление jpg-файла в виде байт-кода после записи нового файла в изображение

Алгоритм записи данных. Рассмотрим программный код, записывающий данные в jpg-изображение (рисунок 4). Весь процесс обзора кода (алгоритма) будет производиться путем последовательного описания каждой его строки.

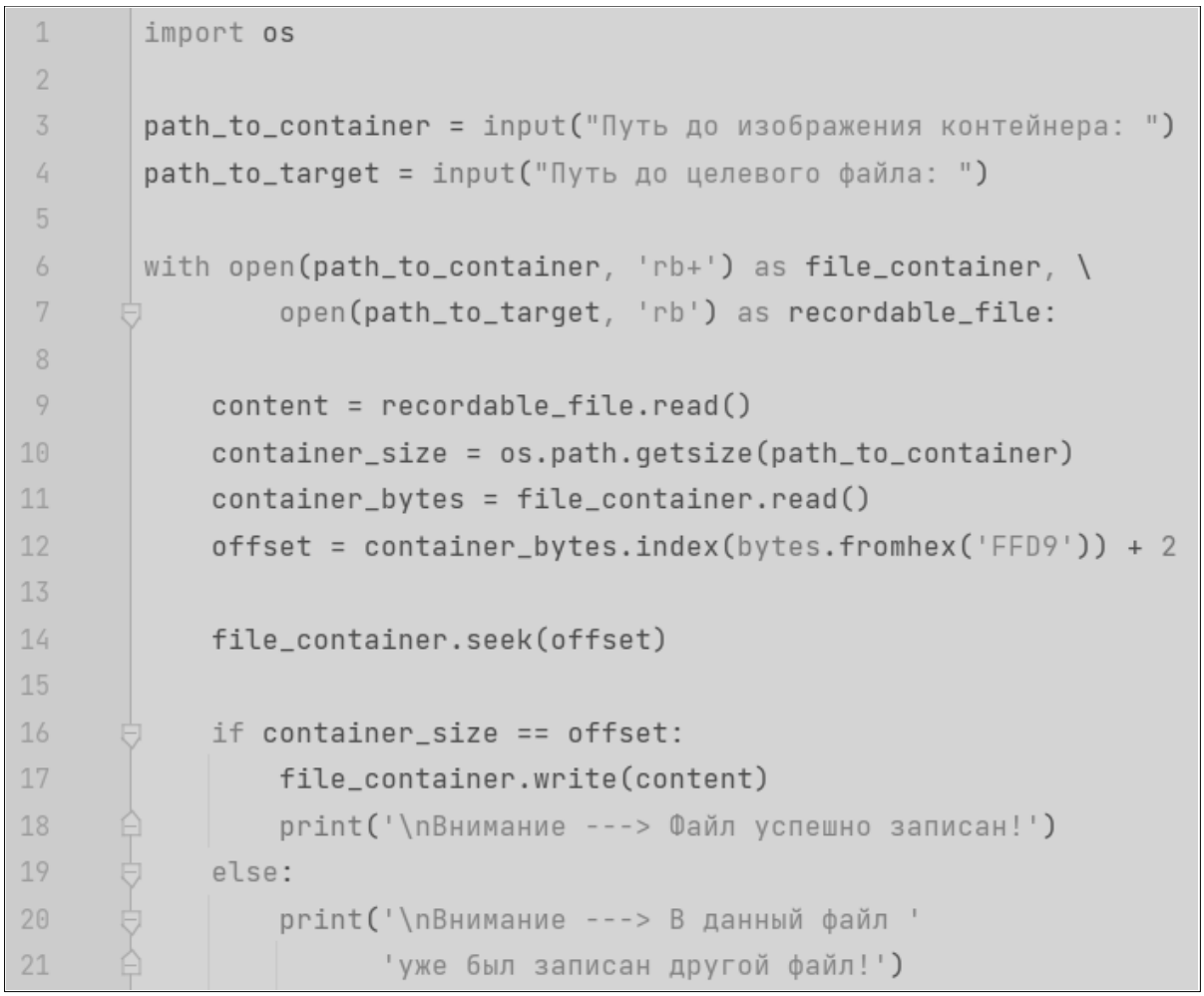
Введем два понятия, которыми будем оперировать в рамках написанного программного кода:

Изображение-контейнер – это jpeg-файл, в который будет дописан сторонний байт-код.

Целевой файл – это файл, байт-код которого будет дописан в изображение-контейнер либо же извлечен из него.

Традиционно в любом языке программирования, перед написанием кода производится импорт всех необходимых сторонних библиотек. Рассмотрим алгоритм, реализуемый программным кодом, представленный на рисунке 4.

Строка 1: в проект импортируется модуль os, позволяющий программисту взаимодействовать с операционной системой (создавать/удалять папки и файлы и так далее).



```
import os

path_to_container = input("Путь до изображения контейнера: ")
path_to_target = input("Путь до целевого файла: ")

with open(path_to_container, 'rb+') as file_container, \
        open(path_to_target, 'rb') as recordable_file:

    content = recordable_file.read()
    container_size = os.path.getsize(path_to_container)
    container_bytes = file_container.read()
    offset = container_bytes.index(bytes.fromhex('FFD9')) + 2

    file_container.seek(offset)

    if container_size == offset:
        file_container.write(content)
        print('\nВнимание ---> Файл успешно записан!')
    else:
        print('\nВнимание ---> В данный файл '
              'уже был записан другой файл!')
```

Рисунок 4 – Программный код, записывающий данные в изображение

Строки 3 – 4: реализуется ввод данных конечным пользователем: путь до изображения-контейнера, в который будет производиться запись, и путь до целевого файла, байт-код которого будет записан в изображение. Для достижения ввода данных используется стандартная функция input().

Строки 6 – 7: выполнив сбор всей необходимой для корректной работы алгоритма информации, реализуется открытие этих файлов для дальнейшей работы с их байт-кодом. Для этого используется конструкция with(), которая имеет два аргумента: путь до файла и режим работы. Всего есть три базовых режима работы: w – write – открытие файла на запись, a – append – открытие на запись в конец файла и r – read – открытие на чтение [2]. Модификатор rb означает, что выбранные данные будут открыты на чтение байт-кода, он расшифровывается, как read bytes – чтение байт. Модификатор rb+ – это режим доступа, при котором происходит открытие на одновременное чтение и запись

байт-кода [3]. Открытым файлам присваиваются псевдонимы, с которыми и будет производиться все дальнейшие манипуляции [2].

Строка 9: производится считывание байтов целевого файла в специальную переменную content.

Необходимо проверить изображение-контейнера на наличие уже дописанной последовательности байт. Внедрение этой проверки позволит избежать ошибок при повторной записи, а также позволит убедиться, что внутри контейнера есть сторонние данные. Для этой цели, был произведен подсчет объема изображения-контейнера двумя способами.

Строка 10: подсчитывается общий объем файла с помощью библиотеки os (первый способ) [4].

Строки 11 – 12: подсчитывается объем изображения строго до маркера «FF D9», обозначающего его конец, (второй способ).

Строки 16 – 18: в случае, если объемы изображения совпадают, будет произведена запись и вывод сообщения об успешности операции.

Строки 19 – 21: если вычисленные объемы не совпали, то будет выведено сообщение о наличие инородного байт-кода в изображении-контейнере.

Подобно работе любого текстового редактора у пользователя есть курсор, который отображает текущую позицию в тексте. Такого рода инструмент используется и при открытии файла на чтение и/или запись. По этой причине перед непосредственной реализацией проверки (строки 16 – 21), требуется выполнить операцию перехода в конец jpg-изображения (строка 14). Поскольку ранее в статье было сказано, что jpg-файл сохраняет свое рабочее состояние только при дописывании в конце.

Алгоритм извлечения данных. Рассмотрим алгоритм, позволяющий считать данные из jpeg-изображения. Программный код алгоритма представлен на рисунке 5. Поскольку алгоритмы записи и считывания в общих чертах схожи и оперируют одними и теми же функциями, рассмотрим его более кратко. Все дальнейшие приведенные ссылки на строки относятся к рисунку 5.

```
1   import os
2
3   path_to_container = input("Путь до изображения контейнера: ")
4   filepath_to_read_into = input("Путь до папки нового файла: ")
5   filename = input("Имя нового файла с расширением: ")
6
7   with open(path_to_container, 'rb') as file_container:
8
9       container_full_size = os.path.getsize(path_to_container)
10      container_bytes = file_container.read()
11      jpeg_file_size = container_bytes.index(bytes.fromhex('FFD9')) + 2
12
13      file_container.seek(jpeg_file_size)
14
15      if container_full_size > jpeg_file_size:
16          with open(filepath_to_read_into + '/' + filename, 'wb')\
17                  as extracted_file:
18
19              extracted_file.write(file_container.read())
20              print('\nВнимание ---> Файл успешно выгружен из jpg-файла!')
21      else:
22          print('\nВнимание ---> В данный jpg-файл не вшит не один файл!')
```

Рисунок 5 – Программный код, извлекающий данные из изображения

Строка 1: производится импорт библиотеки os [4], которая потребуется для проверки наличия данных вшитых в файл-контейнер.

Строки 3 – 5: инициируется сбор данных, необходимых для корректной работы алгоритма, но в отличие от кода записи, передача информации о целевом файле разбита на две части: путь до файла и его имя, записанное вместе с расширением (например, Extracted_file.txt).

Строка 7: инициируется открытие изображения-контейнера в режиме чтение байт-кода [2].

Строки 9 – 11: подсчитываются объемы изображения в целом и до маркера «FF D9», которые в последствие потребуются для проверки наличия данных вшитых в файл-контейнер.

Строка 13: производится перенос курсора (внутренней позиции в байт-коде файла) в конец изображения за счет знания маркера «FF D9» для дальнейшего считывания данных.

Строки 13 – 21: реализуется проверка на наличие данных внутри jpeg-файла.

В случае выполнения проверки открывается новый файл на запись (строки 16 – 17), путь и имя которого основываются на ранее полученных данных (строки 4 – 5). Далее инициируется запись целевого файла и вывод сообщения об успешности операции (строки 19 – 20).

Если же проверка не выполнилась, реализуется вывод соответствующего сообщения (строка 22).

Заключение. Разработанный метод защиты построен по принципам стеганографии, науки, изучающей методы сокрытия и передачи информации, и может быть применен не только для защиты данных, но и для ее скрытой передачи. На основе представленного в статье оригинального алгоритма может быть разработана комплексная система защиты, сочетающая в себе как использование изображений формата jpeg, так и методы криптографической защиты данных (шифр Эль-Гамаля и т. д.).

Библиографический список:

1. Декодируем JPEG-изображение с помощью Python [Электронный ресурс] URL: https://temofeev.ru/info/articles/dekodiruem-jpeg-izobrazhenie-s-pomoshchyu-python/?ysclid=l4ya67aobd854598924/ (дата обращения: 25.06.2022).

2. Writing to file in Python [Электронный ресурс] URL: https //www.geeksforgeeks.org/writing-to-file-in-python/ (дата обращения: 27.06.2022).

3. Python File I/O – Read and Write Files [Электронный ресурс] URL: https //www.tutorialsteacher.com/python/python-read-write-file/ (дата обращения: 27.06.2022).

4. os – Miscellaneous operating system interfaces [Электронный ресурс] URL: https://docs.python.org/3/library/os.html (дата обращения: 23.06.2022).