УДК 004

Извлечение метаданных EXIF с помощью языка программирования Python

Кизянов Антон Олегович
Приамурский государственный университет имени Шолом-Алейхема
Студент

Аннотация
В данной статье рассказано, что такое метаданные, как их получить и чем они могут навредить, если попадут в руки злоумышленника.
Ключевые слова: Python, EXIF, JPEG

Extract EXIF metadata using the Python programming language

Kizyanov Anton Olegovich
Sholom-Aleichem Priamursky State University
student

Abstract
This article tells you what metadata is, how to get it and what they can do if you fall into the hands of an attacker.
Keywords: Python, EXIF, JPEG

Когда человек делает фотографию на телефон, фотоаппарат или другое устройство, он не задумывается о том, что эта фотография может хранить в себе дополнительные данные, вплоть до точного местоположения, где был сделан снимок. А после он публикует их в социальных сетях и любой другой человек может получить эти данные.

Цель исследования – это написание программы для извлечения EXIF метаданных из фотографий на языке программирования Python.

Ранее этим вопросом интересовался К.О.Смирнов развивал тему «Исследование цифровых изображений с помощью EXIF-стандарта» [1] в которой раскрывается вопрос цифрового изображения как объекта судебно фототехнической экспертизы. А.А.Малюка с темой «Цифровое изображение как объект судебной фототехнической экспертизы» [2], а подробнее про цифровые фотографии, которые могут рассматриваться как реальные доказательства в уголовном процессе. Д.А.Токарев, В.Г.Грибунин, И.А.Игнатенко опубликовали статью «Анализ методов уменьшения размеров изображений» [3] рассказали, как работают методы с последующей реализаций методов в некоторых известных пакетах программ.

Метаданные EXIF является стандартом, используемым для изображений и аудиофайлов, созданных устройствами и приложениями.

Чаще всего такие встроенные метаданные связаны с файлами JPEG. Однако метаданные EXIF также присутствуют в файлах TIFF, WAV и других файлах. В файлах JPEG в метаданных EXIF могут содержаться технические параметры камеры, используемые для съемки.

Это может быть бесполезным, но теги, содержащие местоположение фотографии по GPS, могут быть полезны для определения личности. Метаданные EXIF хранятся в начале изображений JPEG. Заголовок EXIF начинается с шестнадцатеричного кода 0x45786966, который кодируется как «Exif» в ASCII.

Можно использовать struct и анализировать через заголовок файла и захватывать соответствующие метаданные EXIF. К счастью, сторонняя библиотека изображений Python, PIL уже поддерживает метаданные EXIF и делает эту задачу намного проще.

Pillow (версия 1.1.7) является активно поддерживаемой веткой библиотеки изображений Python. PIL является обширным модулем, который может архивировать, отображать и обрабатывать файлы изображений. Чтобы установить эту библиотеку, нужно ввести следующую команду:

```
pip install pillow
```

PIL имеет функцию с именем getexif(), которая возвращает словарь тегов и их значений. Теги сохраняются в десятичном формате, а не в шестнадцатеричном формате. Интерпретация 0x010F в big endian соответствует десятичному значению 271 для тега «Make». Вместо того, чтобы делать это через struct, лучшим решением будет просто запросить, существует ли тег и, обработать значение:

```
from PIL import Image
image = Image.open('img_42.jpg')
exif = image._getexif()
if 271 in exif.keys():
    print('Make:', exif[271])
```

Сначала будет рассказано об отдельных функциях, впоследствии они составят готовую программу. Есть три функции: exifParser(), getTags() и dmsToDecimal().

- exifParser() функция, является точкой входа и принимает строку, представляющую путь до изображения.
- getTags() функция отвечает за разбор тегов EXIF из входного файла.
- dmsToDecimal() функция представляет собой небольшую вспомогательную функцию, отвечающую за преобразование координат GPS в десятичный формат.

```
from datetime import datetime
import os
from time import gmtime, strftime

from PIL import Image

import processors

def exifParser():
    …
def getTags():
    …
def dmsToDecimal():
    …
```

Это главный файл где будут прописаны все 3 функции.

Проверка подписи файла, иногда называемая его магическим числом, обычно состоит в том, чтобы исследовать первые пару байтов файла и сравнить это с известными сигнатурами для этого типа файла.

```
def exifParser(filename):
    """
    Функция exifParser подтверждает тип файла и отправляет его для обработки.
    : param имя_файла: имя файла, потенциально содержащего метаданные EXIF.
    : return: словарь из getTags, содержащий встроенные метаданные EXIF.
    """
```

В ней создается список известных сигнатур файлов для изображений JPEG. Вызывается функция checkHeader() из модуля processors. Эта функция будет оценивать, соответствует ли заголовок файла одной из предоставленной известной подписи:

```
    signatures  =  ['ffd8ffdb','ffd8ffe0',  'ffd8ffe1',  'ffd8ffe2',  'ffd8ffe3',
'ffd8ffe8']
    if processors.utility.checkHeader(filename, signatures, 4) == True:
        return getTags(filename)
    else:
        print 'Файловая подпись не соответствует известным подписям JPEG.'
        raise TypeError('Файловая подпись не соответствует объекту JPEG.')
```

Если это действительно файл JPEG, возвращается функция getTags(). Если checkHeader() возвращает False, то вызывается исключение TypeError.

getTags() функция, которая с помощь PIL модуля, анализирует теги метаданных EXIF из изображения. Потом создается список заголовков для CSV-файла. Этот список содержит все возможные ключи, которые могут быть созданы в словаре EXIF, в том порядке, в каком необходимо, чтобы они отображались в файле CSV. Поскольку у всех изображений JPEG могут

быть не те же самые или любые внедренные теги EXIF, может быть так, что одни словари имеют больше тегов, чем другие.

```
        def getTags(filename):
        """
        Функция getTags извлекает метаданные EXIF из объекта данных.
        : param имя_файла: путь и имя объекта данных.
        : return: теги и заголовки, теги - это словарь, содержащий метаданные EXIF и
заголовки.
        порядок ключей для выхода CSV.
        """
        headers = ['Path', 'Name', 'Size', 'Filesystem CTime', 'Filesystem MTime',
'Original Date', 'Digitized Date', 'Make', 'Model', 'Software', 'Latitude',
'Latitude Reference', 'Longitude', 'Longitude Reference', 'Exif Version', 'Height',
'Width', 'Flash', 'Scene Type']
```

Программа откроет файл JPEG, используя Image.open() функцию. Еще раз, используя функцию verify(), выполнит проверку. Эта функция проверяет наличие искажений файлов и вызывает ошибки при обнаружении. Если ошибок нет, вызывается функция getexif(), которая возвращает словарь метаданных EXIF.

```
        image = Image.open(filename)
        image.verify()
        exif = image._getexif()
```

Создается словарь tags, который будет хранить метаданные из фотографии. Заполняется словарь некоторыми метаданными файловой системы, такими как полный путь, имя, размер, а также время создания и изменения. Функция os.path.basename() принимает полный путь к файлу и возвращает имя файла.

Использование getsize() функции вернет размер файла в байтах. Люди привыкли видеть размеры с общими префиксами, такими как MB, GB и TB. Функция процессора convertSize() делает именно это, чтобы сделать данные более понятными.

Преобразование целого числа, возвращаемое от os.path.getctime(), дает время создания, выраженное в секундах с начала эпохи. Эпоха начинается с 01/01/1970 00:00:00. Используется функция gmtime() для преобразования секунд в объект, структурированный по времени (аналогичный datetime).

```
        tags = {}
        tags['Path'] = filename
        tags['Name'] = os.path.basename(filename)
        tags['Size'] = processors.utility.convertSize(os.path.getsize(filename))
        tags['Filesystem CTime'] = strftime('%m/%d/%Y %H:%M:%S',
gmtime(os.path.getctime(filename)))
        tags['Filesystem MTime'] = strftime('%m/%d/%Y %H:%M:%S',
gmtime(os.path.getmtime(filename)))
```

Проверка наличия в exif словаре каких-либо ключей. Если есть, перебирается каждый ключ и проверяется его значение. Значения, которые были запрошены, относятся к тегам EXIF. Существует много потенциальных тегов EXIF, но в этом примере будут запрошены только некоторые из наиболее важных.

Если конкретный тег существует в exif словаре, то он переносится в tags словарь. Некоторые теги требуют дополнительной обработки, например, метки времени, сцены, вспышки и GPS-тегов. Теги метки времени отображаются в формате, который несовместим с тем, как представляется другие временные метки. Например, время от тега 36867 разделяется двоеточиями и в другом порядке.

```
2015:11:11 10:32:15
```

Используется функция для преобразования текущей временной строки в datetime объект. В следующей строке используется функция strftime, чтобы преобразовать ее в нужный формат строки даты:

```
if exif:
    for tag in exif.keys():
        if tag == 36864:
            tags['Exif Version'] = exif[tag]
        elif tag == 36867:
            dt = datetime.strptime(exif[tag], '%Y:%m:%d %H:%M:%S')
            tags['Original Date'] = dt.strftime('%m/%d/%Y %H:%M:%S')
        elif tag == 36868:
            dt = datetime.strptime(exif[tag], '%Y:%m:%d %H:%M:%S')
            tags['Digitized Date'] = dt.strftime('%m/%d/%Y %H:%M:%S')
```

Сцены (41990) и флэш (37385) имеют целое значение, а не строку. Как упоминалось ранее, что представляют собой целые числа. В этих двух сценариях создается словарь, содержащий потенциальные целые числа в качестве ключей, и их описания как значения. Проверяется, является ли значение тега ключевым в словаре. Если он присутствует, то сохраняется описание в tags словаре. Просмотр строкового объяснения сцены или флеш-метки более ценно, чем число:

```
        elif tag == 41990:
            scenes = {0: 'Standard', 1: 'Landscape', 2: 'Portrait', 3: 'Night Scene'}
            if exif[tag] in scenes:
                tags['Scene Type'] = scenes[exif[tag]]
            else:
                pass
            elif tag == 37385:
        flash = {0: 'Вспышка не срабатывала ', 1: 'Вспышка', 5: 'Световой сигнал
строба не обнаружен ', 7: 'Обнаружен обратный свет строба', 9: 'Вспышка, режим
принудительной вспышки', 13: 'Вспышка, принудительный режим вспышки, обратный свет
```

```
не обнаружен', 15: 'Вспышка, режим принудительной вспышки, обнаружен обратный свет', 16: 'Вспышка не срабатывала, режим принудительной вспышки ', 24: 'Вспышка не срабатывала, автоматический режим', 25: 'Вспышка, автоматический режим', 29: 'Вспышка, автоматический режим, обратный свет не обнаружен ', 31: ' Вспышка, автоматический режим, обнаруженный обратный свет ', 32: 'Нет функции вспышки', 65: 'Вспышка, режим уменьшения эффекта красных глаз', 69: 'Вспышка, режим уменьшения эффекта красных глаз, обратный свет не обнаружен', 71: 'Вспышка, режим уменьшения эффекта красных глаз, обнаружен обратный свет', 73: 'Вспышка, режим принудительной вспышки, режим уменьшения эффекта красных глаз', 77: ' Вспышка, принудительный режим вспышки, режим уменьшения эффекта красных глаз, обратный свет не обнаружен', 79: 'Вспышка, режим принудительной вспышки, режим уменьшения эффекта красных глаз, обнаружен обратный свет', 89: 'Вспышка, автоматический режим, режим уменьшения эффекта красных глаз', 93: 'Вспышка, автоматический режим, обратный свет не обнаружен, режим уменьшения эффекта красных глаз', 95: 'Вспышка, автоматический режим, обнаруженный обратный свет, режим уменьшения эффекта красных глаз'}
          if exif[tag] in flash:
              tags['Flash'] = flash[exif[tag]]
          elif tag == 271:
              tags['Make'] = exif[tag]
          elif tag == 272:
              tags['Model'] = exif[tag]
          elif tag == 305:
              tags['Software'] = exif[tag]
          elif tag == 40962:
              tags['Width'] = exif[tag]
          elif tag == 40963:
              tags['Height'] = exif[tag]
```

Тегов GPS, которые хранятся в виде вложенного словаря под ключом 34853 Если существуют теги широты и долготы, они передаются функции dmsToDecimal(), чтобы их преобразовать.

```
      elif tag == 34853:
          for gps in exif[tag]:
              if gps == 1:
                  tags['Latitude Reference'] = exif[tag][gps]
              elif gps == 2:
                  tags['Latitude'] = dmsToDecimal(exif[tag][gps])
              elif gps == 3:
                  tags['Longitude Reference'] = exif[tag][gps]
              elif gps == 4:
                  tags['Longitude'] = dmsToDecimal(exif[tag][gps])
              else:
                  pass
          return tags, headers
```

dmsToDecimal() функция преобразует координаты GPS. Существует простая формула для преобразования между двумя форматами. Данные GPS, которые были извлечены из метаданных EXIF, содержат три кортежа в другом кортеже. Каждый внутренний набор представляет собой числитель и знаменатель степени, минуты или секунды. Во-первых, нужно отделить отдельные градусы, мин и секунды.

На рисунке 1 показано, как можно преобразовать полученные данные GPS в десятичный формат:

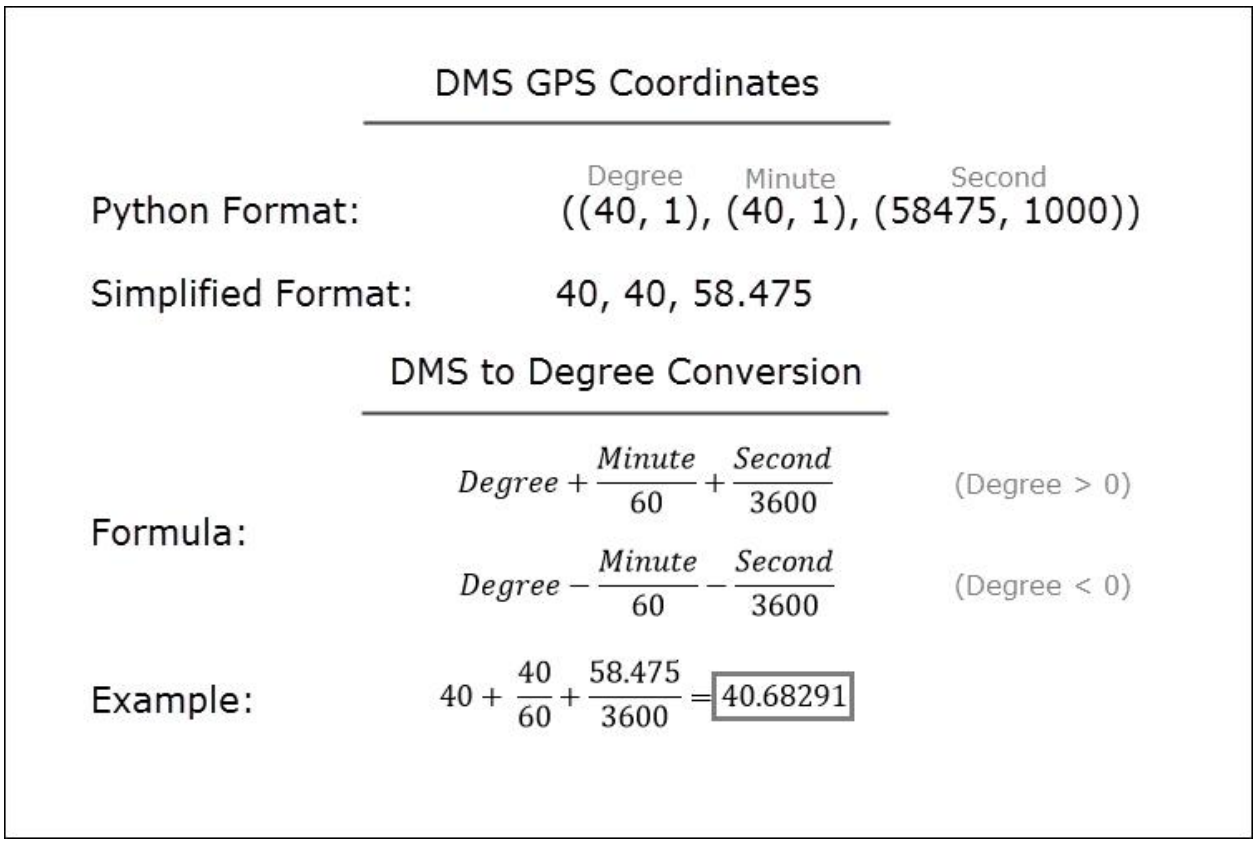

Рис. 1

Используется список для создания списка, содержащего первый элемент каждого элемента в кортеже. Затем распаковывается этот список на три элемента: deg, min, и sec. Используемая формула зависит от того, является ли значение степени положительным или отрицательным.

Если deg положительно, добавляются минуты и секунды. Делятся секунды на 3600000, а не на 3600. Если deg отрицательный, вычитаются минуты и секунды следующим образом:

```
def dmsToDecimal(dms):
"""
:param dms: данные GPS в формате Degree Minute Seconds.
:return: десятичная форматированная GPS-координата."""
    deg, min, sec = [x[0] for x in dms]
    if deg > 0:
        return "{0:.5f}".format(deg + (min / 60.) + (sec / 3600000.))
    else:
        return "{0:.5f}".format(deg - (min / 60.) - (sec / 3600000.))
```

Вывод

Таким образом, можно по одной лишь фотографии человека получить столько данных, что найти его не составит труда. И так как сейчас век информационных технологий пренебрежение такими вещами может плохо кончится.

Библиографический список

1. Смирнов К. О. Исследование цифровых изображений с помощью EXIF-стандарта // Интерактивная наука. 2017. №2. С. 243-246. URL: https://elibrary.ru/item.asp?id=28995624 (Дата обращения: 23.01.2018)
2. Малюка А.А. Цифровое изображение как объект судебной фототехнической экспертизы // Актуальные проблемы российского права 2010. №2А С. 431-441. URL: https://elibrary.ru/item.asp?id=16540786 (Дата обращения: 23.01.2018)
3. Токарев Д. А., Грибунин В. Г., Игнатенко И. А. Анализ методов уменьшения размеров изображений // Известия института инженерной физики. 2014. №31. С. 40-46. URL: https://elibrary.ru/item.asp?id=21223561 (Дата обращения: 23.01.2018)