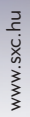
ZABAWY ZE STEGANOGRAFIĄ

PAWEŁ KORUS



W czasach, kiedy prywatność staje się pojęciem coraz bardziej abstrakcyjnym, często potrzebujemy pewnej metody szyfrowania bądź wręcz zatajenia naszej komunikacji. Z pomocą przychodzi steganografia. Najprostsze ze stosowanych początkowo sposobów bazowały na ukrywaniu tekstu w innym tekście poprzez określenie podzbioru liter składających się na właściwą wiadomość. Obecnie, wraz z rozwojem cyfrowych technik kodowania danych, możliwe staje się umieszczenie informacji w zupełnie innym typie danych. Szczególnie użyteczne są dane multimedialne, których odbiór jest warunkowany niedoskonałością ludzkich zmysłów. Stanowi ona podstawę technik kompresji stratnej (na przykład JPEG dla obrazów oraz MP3 dla audio), a odpowiednie jej wykorzystanie pozwala ukryć dane w wybranym dziele w sposób efektywny.

Artykuł omawia zagadnienie ukrywania wiadomości tekstowej w obrazach na przykładzie bitmapy o jednym, 8-bitowym kanale. Zaprezentowana technika może jednak zostać bez trudu rozszerzona do pracy przy innych parametrach. Do implementacji kodera i dekodera wiadomości wykorzystamy język Python oraz biblioteki OpenCV. Efektem będą dwa narzędzia przeznaczone do użycia z linii poleceń.

Przedstawiamy OpenCV

OpenCV to biblioteka programistyczna opracowana przez firmę Intel, która udostępnia szereg zaawansowanych funkcji z dziedziny przetwarzania obrazów. Potencjalne zastosowanie może znaleźć między innymi w projektach potrzebujących funkcji śledzenia ruchu, wykrywania oraz rozpoznawania wzorców (na przykład twarzy), sieci neuronowych oraz wielu innych. Należy podkreślić, że jest udostępniana na licencji BSD, co oznacza, że jest darmowa także do zastosowań komercyjnych. Ostatnie stabilne wydanie nosi numer 1.0.

Biblioteka OpenCV jest przeznaczona głównie dla języków C/C++, jednak standardowo udostępniany jest także interfejs programistyczny dla języka Python. Nie zawiera on wprawdzie jeszcze kompletu funkcji, ale z powodzeniem może zostać wykorzystany w większości przypadków. W połączeniu z wykorzystaniem linii poleceń tego języka dostajemy bardzo użyteczne narzędzie do eksperymentów.

Instalacja nie wymaga żadnych skomplikowanych zabiegów. Dla dystrybucji takich jak Ubuntu czy openSUSE można znaleźć gotowe pakiety, których instalacja nikomu nie przysporzy trudności. Powinniśmy jedynie pamiętać, że w takiej sytuacji API dla Pythona występuje najczęściej w postaci osobnego pakietu. W przypadku braku gotowych paczek dla danej dystrybucji powinniśmy zainstalować OpenCV ze źródeł. W tym celu wystarczy wykonać standardowo

```
./configure
make
make install
make check
```

w katalogu ze źródłami. Zwróćmy uwagę, że na zakończenie procesu konfiguracji zostanie wyświetlone podsumowanie z informacją o tym, które komponenty mogą zostać skompilowane. W razie potrzeby powinniśmy uzupełnić system o odpowiednie biblioteki i pliki nagłówkowe. Jeśli korzystamy z ustawień domyślnych, przed wydaniem polecenia instalacji upewnijmy się, że posiadamy uprawnienia roota. Ostatnia z powyższych komend wykona przygotowane wcześniej testy i sprawdzi, czy wszystko funkcjonuje, jak należy.

Zabawa w chowanego

Gdy mamy już wybrany obraz, którym chcemy się posłużyć, oraz wiadomość, którą chcemy ukryć, powstają pytania: gdzie ją schować? i jak to zrobić, żeby nikt nie widział? Odpowiedzi jest bardzo wiele, my skorzystamy z najprostszej możliwości. Na zapis każdego z pikseli mamy do dyspozycji 8 bitów o rosnących wykładniczo wagach. Przekłada się to na możliwość zakodowania 256 poziomów skali szarości. Jeśli najmłodszy z bitów (o wadze 1) przeznaczymy na zapis naszych danych, zmodyfikujemy intensywność danego piksela w sposób niezauważalny. Rodzinę technik wykorzystującą ten sposób zapisu określa skrót LSB (Least Significant Bit).

W efekcie do dyspozycji mamy wysokość × szerokość bitów na zapis własnych danych. Oznacza to, że wykorzystując obraz o typowych rozmiarach 1280 × 1024 pikseli, dostajemy około 160 KB danych. Pojedynczy znak kodujemy, zgodnie z ogólnie przyjętą metodą, za pomocą 8 bitów. Jest to niewielkie uproszczenie, gdyż w praktyce wiadomości tekstowe najczęściej składają się zaledwie z podzbioru dostępnych znaków i dodatkowych bitów można użyć w innym celu lub wręcz całkowicie je pominąć. Aby przykład był czytelny, zostaniemy przy prostej konwersji znaku na 8-bitową liczbę.

Ponieważ chcemy mieć możliwość ukrywania wiadomości o nieznanej wcześniej dłu-

gości, musimy poinformować dekoder, w którym miejscu zakończyć przetwarzanie danych. W tym celu wykorzystamy znak o kodzie 0xFF. Nie stanowi to problemu, gdyż w praktyce, znak ten nie pojawia się w żadnych wiadomościach tekstowych.

Pozostaje jedynie kwestia wyboru współrzędnych pikseli, które mają zostać użyte. W tym przypadku posłużymy się generatorem liczb pseudolosowych, który będzie dokonywał wyboru pikseli z puli tych jeszcze nie wykorzystanych. Dzięki temu nasza wiadomość jest dodatkowo zabezpieczona kluczem w postaci liczby wykorzystywanej do inicjalizacji generatora. Oczywiście nadawca oraz odbiorca muszą uzgodnić w sposób bezpieczny klucz przed rozpoczęciem wymiany ukrytych wiadomości.

Listing 1: Kodowanie kolejnych bitów w pikselach obrazu

```
for m_bit in message_bits:
    pos = pix[random.randint(0, len(pix)-1)]
    pix.remove(pos)
    cr, cc = pos/img.width, pos - (pos/img.width)*img.width
    img[cr, cc] = (img[cr, cc] & 0xFE) + m_bit
```

Rysunek 1: Ilustracja skali zmian wprowadzonych przez ukrycie wiadomości w obrazie. Od lewej: obraz oryginalny, obraz zmieniony, ilustracja miejsc zmian.

Zabieramy się do pracy

Powyższy schemat pozostaje zapisać w postaci kodu. Poznamy tu elementarne operacje potrzebne do implementacji zasadniczych funkcji, aby przyjrzeć się wszystkim etapom kodowania oraz dekodowania krok po kroku. Dzięki temu zapoznamy się z podstawami pracy z OpenCV. Gotowy kod źródłowy narzędzi (uzupełniony o obsługę parametrów linii poleceń) dostępny jest na stronie internetowej „Linux Magazine" [1]. Plik *encode-lsb.py* jest narzędziem trybu tekstowego służącym do kodowania wiadomości. Program dekodujący zawarty jest w pliku *decode-lsb.py*.

Tradycyjnie rozpocząć musimy od importu stosownych modułów:

```
import sys
import random
from opencv import *
from opencv.highgui import *
```

Moduł *opencv* udostępnia funkcje zajmujące się właściwym przetwarzaniem macierzy i obrazów. Znajduje on zastosowanie we wszystkich aplikacjach opartych na omawianych bibliotekach. Natomiast *highgui* zawiera funkcje przydatne głównie w prostych, eksperymentalnych programach (na przykład tworzenie prostych okien i wyświetlenie w nich obrazu).

Pierwszą operacją, jaką należy wykonać, jest wczytanie bitmapy, w której zamierzamy ukryć wiadomość. Wykonujemy to za pomocą funkcji

```
mg = cvLoadImage(path,
CV_LOAD_IMAGE_GRAYSCALE)
```

Podczas wczytywania obrazu mamy możliwość określenia dodatkowych flag. W tym przypadku skorzystaliśmy z *CV_LOAD_IMAGE_GRAYSCALE*, co powoduje, że wskazana bitmapa zostanie automatycznie przekonwertowana do skali szarości. Oszczędzamy dzięki temu czas i pamięć.

Na tym etapie należy sprawdzić, czy zadany obraz został poprawnie wczytany oraz czy posiada wystarczającą pojemność dla naszej wiadomości. Do rozmiarów obrazu możemy się odwołać poprzez atrybuty *width* i *heigh* oraz *rows* i *cols* utworzonego obiektu. Wynika to z faktu, że bitmapy w OpenCV są traktowane jako rozszerzenie pojęcia macierzy. Pamiętamy, że do zakodowanego tekstu należy dodać jeden znak (8 bitów) końca pracy dekodera. Jeżeli powyższy warunek jest spełniony, przystępujemy do inicjalizacji generatora liczb pseudolosowych za pomocą predefiniowanego klucza (*seed*).

```
if not img:
    sys.exit(2)
message += chr(0xFF)
if 8*len(message) > img.width*img.height:
    sys.exit(3)
random.seed(seed)
```

Uzupełnioną o dodatkowy znak wiadomość zamieniamy na ciąg bitów, co nie powinno sprawić problemów. Następnie generujemy listę dostępnych pikseli w postaci kolejnych liczb, począwszy od 0:

```
pix = range(0, img.width*img.height)
```

Odpowiadające tym liczbom współrzędne możemy uzyskać z zależności:

```
cr = pos/img.width
cc = pos - (pos/img.width)*img.width
```

przy czym *pos* oznacza numer wylosowanego piksela, *cr* – aktualny wiersz, a *cc* – aktualną kolumnę macierzy.

Teraz pozostaje już tylko przeprowadzić iterację po kolejnych bitach kodowanej wiadomości poprzez nadpisanie najmniej znaczącego bitu wylosowanych pikseli. Operator *[]* daje nam dostęp do stosownych wierszy bądź elementów macierzy – w zależności od podanej liczby indeksów. Zatem ostatecznie operacja nadpisania najmłodszego bitu przedstawia się jak na Listingu 1.

Aby zapisać powstały obraz, wykonujemy polecenie

```
cvSaveImage(out_path, img)
```

Format zapisu zostanie automatycznie wybrany na podstawie podanej nazwy pliku. Należy pamiętać, że przedstawiona technika wymaga użycia takiego formatu, który nie korzysta z kompresji stratnej. Dobrym wyborem jest przykładowo PNG oferujący kompresję bezstratną.

Listing 2: Gotowa implementacja głównej pętli dekodera

```
buffer = []
message = []
decode = True

while decode and len(pix) > 0:
  pos = pix[random.randint(0, len(pix)-1)]
  available_positions.remove(pos)
  cr = pos/img.width
  cc = pos - (pos/img.width)*img.width
  buffer.append(img[cr, cc] & 0x01)

   if len(buffer) == 8:
     val = 0
    for m_bit, v in zip(buffer, range(8)):
          val += pow(2, v)*m_bit

        buffer = []

        if val == 0xFF:
            decode = False
        else:
            message.append(chr(val))
```

Rysunek 2: Porównanie histogramów obrazu przed i po modyfikacji (odpowiednio po lewej i prawej stronie).

Procedura odczytania ukrytej w ten sposób wiadomości przebiega w sposób odwrotny. Początkowe operacje – wczytanie obrazu, inicjalizacja generatora liczb pseudolosowych oraz utworzenie listy wszystkich pikseli obrazu – pozostają bez zmian. Pętla iterująca po bitach wiadomości zostaje zamieniona na pętlę kontynuującą swoje działanie do momentu napotkania znaku końca wiadomości (0xFF) (bądź wyczerpania pikseli, rzecz jasna). Kolejne bity przechowujemy w buforze, aż do momentu zgromadzenia 8 bitów interpretowanych jako konkretny znak. Ostatecznie uzyskujemy efekt jak na Listingu 2.

Przykład zastosowania przedstawionej techniki zilustrowano na Rysunku 1. Przedstawia on kolejno obraz oryginalny, obraz z ukrytą wiadomością oraz ilustrację pikseli, które zostały zmienione podczas zapisu (uzyskany poleceniem *compare* z pakietu ImageMagick). Użyta w tym przykładzie bitmapa miała rozmiar 128 × 128 pikseli. Zakodowana wiadomość to „Hello World!”, co stanowi zaledwie ułamek procenta dostępnej pojemności.

Miarą modyfikacji wprowadzonych do obrazu może być także porównanie histogramów oryginalnego oraz przetworzonego obrazu. Rysunek 2 przedstawia histogramy odnoszące się do powyższego przykładu. Warto nadmienić, że histogramy te zostały wygenerowana również za pomocą OpenCV. Wracając jednak do histogramu, zmiana jest i tu niezauważalna, co potwierdza brak widocznego wpływu na wygląd obrazu przy korzystaniu z opisanej techniki.

Podsumowanie

Zaprezentowana metoda ukrywania wiadomości w obrazach należy do najprostszych. Dzięki temu nie ma wygórowanych wymagań pamięciowych i jest bardzo szybka. Posiada jednak szereg ograniczeń, które sprawiają, że nie nadaje się do pewnych zastosowań. Zasadniczą wadą jest całkowity brak odporności na modyfikacje obrazu. Oznacza to, że potencjalny atak, mający na celu uniemożliwienie odczytania wiadomości, jest prosty do przeprowadzania (na przykład poprzez operację kadrowania).

Dodatkowo należy zauważyć, że główna pętla dekodera przetwarza znaki do momentu napotkania na symbol 0xFF lub do wyczerpania listy dostępnych pikseli. W przypadku, gdy znany jest kod inicjalizujący (*seed*), proces odczytywania wiadomości jest bardzo szybki. W przeciwnym razie analizowane są wszystkie dostępne współrzędne obrazu, a efektem jest stosunkowo długi czas oczekiwania na zupełnie niezrozumiały ciąg znaków. Jeśli jednak osoba postronna ma świadomość faktu ukrycia wiadomości tą techniką i jest w stanie narzucić pewne ograniczenia na kodowane dane (na przykład fakt, że występują jedynie wybrane znaki lub dozwolone są tylko ich wybrane sekwencje), możliwe jest odgadnięcie ukrytej wiadomości za pomocą metody siłowej (*bruteforce*).

Dodatkowo, ponieważ wymagana jest wierna reprezentacja obrazu, nie można wykorzystać do jego zapisu formatu bazującego na kompresji stratnej (na przykład JPEG). Wspomniane problemy zostały już rozwiązane przy użyciu odmiennego, w stosunku do opisanego, podejścia. Posiadają one jednak także swoje ograniczenia. Artykuł miał pokazać, że dysponując tak potężnym narzędziem, jakim jest OpenCV, mamy szeroki wachlarz możliwości, a technika spełniająca nasze wymagania leży w zasięgu ręki… ■

INFO

[1] Kod źródłowy narzędzi: *http://linux-magazine.pl/index.php/issues/62*

[2] *http://sourceforge.net/projects/opencvlibrary*

[3] I.Cox, M.Miller, *Digital Watermarking and Steganography*, Morgan Kaufmann 2008.

[4] *http://www.watermarkingworld.org/*

[5] *http://www.cosy.sbg.ac.at/~pmeerw/Watermarking/*

AUTOR

Paweł Korus jest doktorantem z dyscypliny Telekomunikacja na AGH w Krakowie. Jego zainteresowania naukowe obejmują min. przetwarzanie obrazów oraz cyfrowe znaki wodne.