Multimedia Programming

Steganography

Copyright © Software Carpentry 2010
This work is licensed under the Creative Commons Attribution License
See http://software-carpentry.org/license.html for more information.

Data is 1's and 0's
Data is 1's and 0's
Normally think of them as integers, characters, etc.

Data is 1's and 0's
Normally think of them as integers, characters, etc.
But sometimes useful to go back to the bits
Data is 1's and 0's
Normally think of them as integers, characters, etc.
But sometimes useful to go back to the bits
Example: hide messages in images

Steganography
Multimedia Programming

Steganography

(53, 64, 22)
8 bits each

(53, 64, 22)

8 bits each

56 32 98 105 116 115 32 101 97 99 104

8 bits each
Replace color bytes with character bytes

Main driver

```python
if sys.argv[1] == '-e':
    message = sys.argv[2]
    pic = Image.open(sys.argv[3])
    encode(message, pic)
    pic.save(sys.argv[4])

elif sys.argv[1] == '-d':
    pic = Image.open(sys.argv[2])
    message = decode(pic)
    print message
```
Main driver

```python
if sys.argv[1] == '-e':
    message = sys.argv[2]
    pic = Image.open(sys.argv[3])
    encode(message, pic)
    pic.save(sys.argv[4])

elif sys.argv[1] == '-d':
    pic = Image.open(sys.argv[2])
    message = decode(pic)
    print message
```

$ steg -e 'ABCDEF' in.jpg out.jpg

$ steg -d out.jpg
Main driver

```python
if sys.argv[1] == '-e':
    message = sys.argv[2]
    pic = Image.open(sys.argv[3])
    encode(message, pic)
    pic.save(sys.argv[4])

elif sys.argv[1] == '-d':
    pic = Image.open(sys.argv[2])
    message = decode(pic)
    print(message)
```

Encode

```python
def encode(message, pic):
    assert len(message) < 256, 'Message is too long'
    set_red(pic, 0, 0, len(message))
    i = 1
    for c in message:
        set_red(pic, 0, i, ord(c))
        i += 1
```
Encode

def encode(message, pic):
    assert len(message) < 256, 'Message is too long'
    set_red(pic, 0, 0, len(message))
    i = 1
    for c in message:
        set_red(pic, 0, i, ord(c))
        i += 1
Encode

def encode(message, pic):
    assert len(message) < 256, 'Message is too long'
    set_red(pic, 0, 0, len(message))
    i = 1
    for c in message:
        set_red(pic, 0, i, ord(c))
        i += 1
Encode

def encode(message, pic):
    assert len(message) < 256, 'Message is too long'
    set_red(pic, 0, 0, len(message))
    i = 1
    for c in message:
        set_red(pic, 0, i, ord(c))
        i += 1

def set_red(pic, x, y, val):
    r, g, b = pic.getpixel((x, y))
    pic.putpixel((x, y), (val, g, b))
Encode

def encode(message, pic):
    assert len(message) < 256, 'Message is too long'
    set_red(pic, 0, 0, len(message))
    i = 1
    for c in message:
        set_red(pic, 0, i, ord(c))
        i += 1

def set_red(pic, x, y, val):
    r, g, b = pic.getpixel((x, y))
    pic.putpixel((x, y), (val, g, b))

Decode

def decode(pic):
    num_chars = get_red(pic, 0, 0)
    message = '
    for i in range(1, num_chars+1):
        message += chr(get_red(pic, 0, i))
        i += 1
    return message
Decode

def decode(pic):
    num_chars = get_red(pic, 0, 0)
    message = ''
    for i in range(1, num_chars+1):
        message += chr(get_red(pic, 0, i))
        i += 1
    return message
```python
def decode(pic):
    num_chars = get_red(pic, 0, 0)
    message = ''
    for i in range(1, num_chars+1):
        message += chr(get_red(pic, 0, i))
        i += 1
    return message
```

Why not `str`?
Decode

def decode(pic):
    num_chars = get_red(pic, 0, 0)
    message = ''
    for i in range(1, num_chars+1):
        message += chr(get_red(pic, 0, i))
    return message

def get_red(pic, x, y):
    r, g, b = pic.getpixel((x, y))
    return r

Result

'ABCDEF'
Result

Multimedia Programming

Steganography
Result

'ABCDEF'

Multimedia Programming  Steganography
Result

'ABCDEF'

JPEG is a *lossy* format
JPEG is a *lossy* format
Throw away some information to improve compression

Human eye can't tell the difference...
JPEG is a *lossy* format
Throw away some information to improve compression
Human eye can't tell the difference...
...but uncompressed image is not identical to original

Not very good for hiding messages...
JPEG is a *lossy* format
Throw away some information to improve compression
Human eye can't tell the difference…
…but uncompressed image is not identical to original
Not very good for hiding messages...
Use a *lossless* format like PNG instead

Try program on a square white PNG
Try program on a square white PNG

$ steg -e 'ABCDEF' white.png encoded.png

ValueError: too many values to unpack

def set_red(pic, x, y, val):
    r, g, b = pic.getpixel((x, y))
    pic.putpixel((x, y), (val, g, b))
Try program on a square white PNG

$ steg -e 'ABCDEF' white.png encoded.png

ValueError: too many values to unpack

def set_red(pic, x, y, val):
    r, g, b = pic.getpixel((x, y))
    pic.putpixel((x, y), (val, g, b))

Pixel at (0, 0) is (255, 255, 255, 255)
Try program on a square white PNG

$ steg -e 'ABCDEF' white.png encoded.png

ValueError: too many values to unpack

def set_red(pic, x, y, val):
    r, g, b = pic.getpixel((x, y))
    pic.putpixel((x, y), (val, g, b))

Pixel at (0, 0) is (255, 255, 255, 255)

Easy to fix...

alpha (transparency)

Result

size (6)

characters
Result

Not very well hidden...

Solution: only use the least significant bit of the color in each pixel
Solution: only use the least significant bit of the color in each pixel

Human eye cannot see difference between (140, 37, 200) and (141, 36, 201)

'A' = 65_{10} = 01000001_{2}
Solution: only use the least significant bit of the color in each pixel

Human eye cannot see difference between
(140, 37, 200) and (141, 36, 201)

'A' = 65_{10} = 01000001_{2}

(8 bits/character) / (3 bytes/pixel) = 3 pixels/character

(With one bit unused)
Extract bits

```python
def get_bits(char):
    num = ord(char)
    result = [0] * 8
    for i in range(8):
        if (num % 2) != 0:
            result[i] = 1
        num /= 2
    return result
```
Extract bits

def get_bits(char):
    num = ord(char)
    result = [0] * 8
    for i in range(8):
        if (num % 2) != 0:
            result[i] = 1
            num /= 2
    return result
Extract bits

def get_bits(char):
    num = ord(char)
    result = [0] * 8
    for i in range(8):
        if (num % 2) != 0:
            result[i] = 1
        num /= 2
    return result

char 'A'
num $65_{10} = 01000001_2$
result $[1,0,0,0,0,0,0,0]$
i $0$

char 'A'
num $32_{10} = 00100000_2$
result $[1,0,0,0,0,0,0,0]$
i $1$
Extract bits

```python
def get_bits(char):
    char = 'A'
    num = ord(char)
    result = [0] * 8
    for i in range(8):
        if (num % 2) != 0:
            result[i] = 1
        num /= 2
    return result
```

Extract bits

```python
def get_bits(char):
    char = 'A'
    num = ord(char)
    result = [0] * 8
    for i in range(8):
        if (num % 2) != 0:
            result[i] = 1
        num /= 2
    return result
```
Combine with pixels

def combine(pixel, bits):
    assert len(pixel) == len(bits), 'Length mismatch'
    pixel = list(pixel)
    for i in range(len(pixel)):
        even = 2 * (pixel[i] / 2)
        if bits[i]:
            even += 1
        pixel[i] = even
    return tuple(pixel)
def combine(pixel, bits):
    assert len(pixel) == len(bits), 'Length mismatch'
    pixel = list(pixel)
    for i in range(len(pixel)):
        even = 2 * (pixel[i] / 2)
        if bits[i]:
            even += 1
        pixel[i] = even
    return tuple(pixel)
Test

```python
for (p, b) in (((0, 0, 0), (0, 1, 0)),
               ((1, 1, 1), (1, 0, 1)),
               ((2, 2, 2), (1, 0, 1)),
               ((255, 255, 255), (0, 1, 1))):
    result = combine(p, b)
    print p, '+', b, '=>', result
```

(0, 0, 0) + (0, 1, 0) => (0, 1, 0)
(1, 1, 1) + (1, 0, 1) => (1, 0, 1)
(2, 2, 2) + (1, 0, 1) => (3, 2, 3)
(255, 255, 255) + (0, 1, 1) => (254, 255, 255)

Write the other functions
Write the other functions

Most important message: bits don't mean anything

Meaning comes from how we act on them