STEGANOGRAPHY: Hiding Audio in Image

Objective: To encode an audio file inside an image and subsequently retrieve the same uncorrupted audio file using Steganography as the medium of communication.

TEAM MEMBERS

Nisarg Nitin Thakur

Sagari Raju Vatchavayi

Christianah Adigun

1. Introduction

Information can be concealed in a way that no one, except the desired end user is able to retrieve it. This can be achieved using a technique known as Steganography. The data to be hidden, is embedded into seemingly regular media such as images, audio files, etc. without making a noticeable difference to its inherent properties. A secret algorithm is used to decode it. Steganography prevents unauthorized users to have access to important private data and mainly deals with protecting the contents of a message while transmitting it. A good implementation of a steganography based algorithm must not alter the characteristics of a message such as number of bits, color changes, etc.

In our project, we used an image with a .png extension in which an audio file was embedded. After performing the encoding and decoding, we were able to retrieve the audio without any losses or additional noise.

2. How the tool works

Our tool takes an audio file (.wav) and hides it into an image (.png). The new image is then sent to the receiver and receiver extracts the audio out of the image.

The logic that the tool uses is,

If the sum of the R, G, B values of a pixel is divisible by 3, read the pixel as 1, else read the pixel as 0. So if we have bits '11010001' from the audio to encode and the first pixel reads '1' then we don't have to do anything, skip the pixel. Take the next bit and next pixel, next bit is '1' and if the next pixel reads '0', then we will change the green value of the pixel such that the pixel reads '1'. Similarly all the bits are mapped in the image. We are not saving the actual bits in the image, but we are altering the image such that each pixel will be read as the bit values of the audio.

Algorithm:

1. Encoding:
    a. Take the image and the audio file path to begin with the steganography.
    b. Take the audio file,
        i. Convert the audio file into string.
        ii. Convert the audio string into binary.
        iii. Append a padding at the end of the binary string. //End of audio file
    c. Take the image file,
        i. Until audio file
            1. Get one pixel
            2. Find sum of R, G, B values of that pixel
            3. If the sum is divisible by 3
                a. Set divisible as 1

4. Else
    a. Set divisible as 0
5. If binary bit is 0 and divisible is 1
    a. Change the green bit so that sum is not divisible by 0
6. Else if binary bit is 1 and divisible is 0
    a. Change the green bit so that sum is divisible by 0
7. Else //If the bit and divisible are same
    a. Pass //No need to do anything

ii. Create the new image based on the new pixel data

d. Save the new image file.

2. Decoding:

a. Take the image file

i. Until the padding

1. Take one pixel

a. If sum of R, G, B of the pixel is divisible by 3

i. Append 1 to the binary

b. Else

i. Append 0 to the binary

b. Remove the padding from the binary data

c. Convert the binary data into string

d. Save the new data as 'filename.wav' //Our hidden audio

3. Process and Challenges

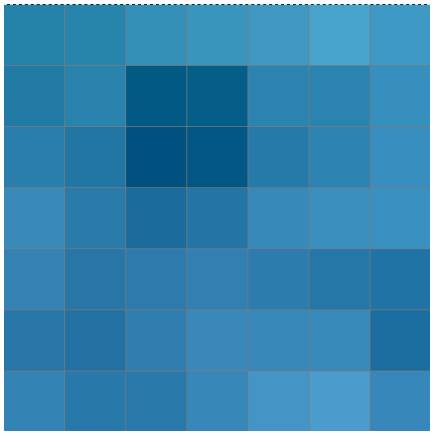
1. Our tool initially stored the binary bits of the audio in the LSB of each pixel, which is the most common technique used in steganography.
2. This technique had many flaws and hence we decided to implement F5 steganographic algorithm, but it was not possible to implement it in such a short duration, as it is very complicated and uses multiple algorithms to scatter the encoded pixels uniformly across the image.
3. Then we had this idea based on a method of how covert channels work,
    - For example, if two people wants to communicate in the system and if they decide to check the write permissions of a file.
    - If the write permissions change, consider it as 1
    - If they don't change, consider it as 0.
4. Similarly, in our tool, the way the audio file is extracted is,
    - If the sum of the R,G,B values of the pixel is divisible by 3, consider that pixel as 1,
    - Else consider that pixel as 0.
5. This is taken into consideration when the image is encoded, as the actual bits are not stored, but the pixels are altered in such a way that they will be read as the bit data of the audio.
6. The main challenge while using this logic was to maintain the integrity of the audio file in the image and extracting the data without any corruption.
7. Many times the padding that was used to indicate the end of the binary bits of the audio was not effective.
8. The pattern would appear in the image before the end of the actual audio bits, resulting into incomplete audio. So with multiple trial and errors, an effective padding was chosen.

9. Also the green value of the pixel was needed to be checked before altering it, if the value of green was 255, adding 1 would result in corrupting the pixel. It also resulted some noise in the audio, and noise in the encoded image which was visible by human eye.
10. Yet, as for now, the limitation of the tool is, the audio file should be less than half of the size of the image, so that the audio is successfully retrieved back.

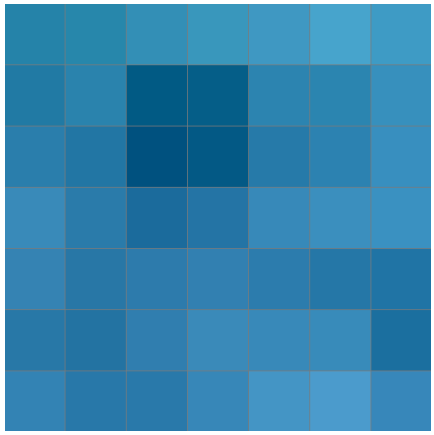
4. Analysis of our tool

The audio-embedded image is completely indistinguishable from the original image by the human eye. The following is a cropped 7 x 7 pixels sample from the original image used to hide an audio message and its corresponding encoded version.

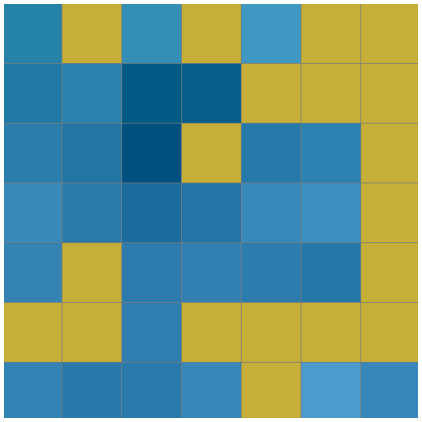
ORIGINAL IMAGE



MODIFIED (AUDIO-ENCODED) IMAGE



As seen above, the differing pixels in are not visible to the human eye. A security engineer will need to have a copy of the original image and the audio-encoded version and analyze each pixel difference. Using special analysis tools, an encoding may be detected in the modified version but it's hard to distinguish a pattern. Below are the marked differing pixels which change by 1 or 2 bytes.



5. Instructions to use the tool

With the package a sample image and an audio file has been provided for convenience; however, if you want to choose a different audio and image files, make sure the audio file is less than half of the size of the image.

1. Extract the package containing code and the sample data
2. Open Terminal
3. Go to the folder where the package is extracted.
4. To Encode
    a. Enter the following line to encode the audio into the image.
        i. python audio.py -e sample_img.png
        ii. It will ask for the filename of the audio.
        iii. Enter sample_audio.wav //Or other audio filename(wav)
    b. A new image file will be generated with the audio encoded in it.
5. Send the encoded image to the receiver.
6. To Decode
    a. Enter the following command into the terminal
        i. python audio.py -d newfile.png
    b. The audio will be extracted from the image named extracted.wav

NOTE: Code may take some time to encode depending upon the size of the audio file and image.