Audio Steganography Instructions

December 16, 2018

Created by David Story for Spring 2019 Workshop

0.0.1 Objective

This workshop is intended to help students who have taken CS115 or an equivalent course. We will be working in Python 3.

0.0.2 Mission

In this workshop, we will be creating a class that will allow us to decode information that was hidden inside an audio file.

Your mission is to be able to process the audio file, extract hidden bits in that file, make each 8 bits a byte that will coorespond to an ASCII value, print the secret message or write it to a file, which will reveal the coordinates to a hidden package.

The location will be within the bounds of the campus. To support any students that are struggling to complete the decoder, code for encoding information in audio files will be released midway through the workshop. You may also ask assistance of the workshop instructors at any time if you have questions.

Groups that can successfully decode the audio and locate the secret will be awarded a certificate of completion on delivery of the objective.

0.0.3 Concepts You Should Know

Print Statements
- Data Types
- String Manipulation
- Logical Statements and Operators
- For, While, Nested Loop Concepts
- Basic File Operations
- Basic Math Operations and Operators
- Single & Multidimensional List Concepts and Operations

0.0.4 Concepts You Will Learn

Bitwise Operations
- Creating and Importing Libraries
- Class Concepts
- Additional File Operations

Basic Steganography Concepts
Basic Data Parsing

0.0.5 What is Steganography?

From the Merriam-Webster Dictionary, Steganography is defined as "The art or practice of concealing a message, image, or file within another message, image, or file". [1]

Steganography can be used to hide information where people least expect it. Audio, images, and video are only a few types of data that you can hide messages in. Decoding the data requires knowing how the information is encoded in order to properly decode the message. Thus, this making steganography a safe and subtle way of hiding information in 'plain sight'.

0.0.6 What you are given

Thus in our case, you will be provided with a folder containing various audio files which already have a text file "hidden" in the bytes of the audio file. You will also be given two Python files: One called AudioParse.py that can take your audio files and return a list of signed 16-bit integers which represent each audio sample. You will also get a file called AudioSteganography that you will add code to.

The framework has comments that shows you what each support function does. These functions are here to remove some of the more tedious parsing and minor issues that can be time consuming. You can open and examine each file after the workshop if you want to know more about whats going on 'under the hood'.

0.0.7 What you are looking for

You will be looking for a message that consists of ASCII characters that form a string that has the name of a location, lattitude and longitudial coordinates, and an additional location description to aid you in finding your secret package. You can use this website to enter the coordinates you are given.

0.0.8 Getting your starting code

You will be given the framework in the form of a Python file, where you will implement the code for a decoding class. The framework will provide a general structure, but you are free to approach the decoding algorithm from whatever angle you think is best. Therer are two support files that I have provided, one being AudioParse.py and the other being AudioStenography.py. These two files provide important code that allow you to turn the .wav file into a list of signed integers. With this list, you can begin to create your decoding program. You can get the following code by downloading it from github or you can get it with the following git command:

git clone https://github.com/david-story/Audio-Steganography.git

0.0.9 Encoding Information

Each of the audio files provided is a 16-bit signed, Mono .wav file. This means that each audio sample in the file can be represented as an integer between the values 32,768 to 32,767.

The message hidden inside the audio files come from a text file that contains a string of ASCII characters. Each characters consists of one byte (8-bits). For this kind of steganography, we take

each character and hide each of the characters bits starting from the most significant bit (MSB) to the least significant bit (LSB) in the audio samples.

As an example, if we have the ASCII character 'a', it's equivalent decimal value is 61, and its equivalent binary representation is: 01100001

Using 'a' as our example byte, starting from left to right (MSB to LSB), we take each bit and we encode it in the least significant bit (LSB) of an audio sample . For simplicity, we are considering the 16-bit signed integer as one 'byte' or our one sample. Thus, it would take 8 audio samples to encode one ASCII character into it.

Since audio files are large, you can typically fit all your message into a portion of the audio. To indicate that the message is done and that you can stop decoding, we include a sentinel byte that is value 0x00. So, once your decoder extracts 8 bits from 8 audio samples and 'sees' that it is a byte of 0x00, your decoding should stop.

0.0.10 Things to think about

Things to consider when making your decoder:

Does the signing of the audio samples matter when extracting its LSB?

How can you read just the LSB of the audio samples?

How many bytes are in your audio sample?

How many bits can be hidden in that sample?

How can you store your bits while you traverse the audio samples?

0.0.11 Begin!

If you haven't already, download the framework and begin coding. Keep in mind how the information is encoded, and please ask for help whenever needed!