Computer NetworksImage Steganography

short line

Nashmia Riaz Reg 33618

Dua Anjum Reg 32120

Saad Raza Reg 32911  
14th May, 2016

Overview:

For our computer networks project, we have made a TCP web server from scratch using socket programming. The user accesses our website by using the web address localhost:55555/index.html, where he can select a message. The message is then encoded in a picture and made available for downloading to the user. The user then runs the client code on the image, which decodes it and displays the message in the web browser.

TCP Web Server:

The web server code works firstly by defining the port, which is set to 55555 as per the specifications of the project. Then, a socket is created, and the port and IP address (which in this case is left blank as to create a localhost connection) is ‘binded’ to it. Next, the listen function is called, where the server has now created a passive open and waits for a connection from the client.

Next, in an endless loop, the following code is followed unless and IO error occurs, in which case it’s simply printed. The connection from client is accepted and packets are then received. These packets contain the URL for the file to be opened, as in our programming in the index.html page, we’ve added links to the different messages. This URL is then extracted, and the file is opened and read. If the user clicks the ‘original image’ link, the original (not encoded) image is simply sent to the user, where it’s downloaded to their browser. Otherwise, if the user opens a message, which is kept in a text file, then that text file is opened. The message is then extracted, and encoded into a copy of the original image.This image is then sent to the client. After the data is sent, the socket is closed as this server is imitating HTTP 1.1, which is non-persistent. If at any point the user opens a file which is not on the server, then a 404 html page is opened.

Encoding:

We used Least significant bit algorithm to encode the message in the pixels of an image. First in the encode function of our code we calculate the length of the message to be encoded. Then we check whether the length of the message is greater than maxsize that can be encoded in the image we consider the length to be the maxsize.   
Now we check if the image which is to used is in the correct mode to be encoded which is ‘RGB’. if it is in RGB mode we continue forward , if not then the program gives an error of ‘incorrect image mode’.   
After checking the image mode we make a copy of that image and encode the message in that copy. We extract the necessary information of the picture like dimensions , size etc.

Then we convert our message into binary so that we can bitwise manipulate the message. Then we run two loops which transverse the whole picture. In the inner loop which is running in a row we will do the encoding.

Firstly we extract the RGB values of the pixels. Then we convert the length of the message from integer to binary. The number of bits in the RGB binary pixels are 18. So when we convert the length of the message to binary it won’t always have 18 bits. So to add necessary number of zeroes in the binary length we will run a loop which will make sure that the length of message in binary is always 18 bits.   
Then we converted the RGB values into binary neglecting the first two bits as they are b and 0 which are not important. After that it takes the rest of the bits till the 2nd last bit of the pixel in binary value and adds the bit of the message length to each pixel (RGB). it keeps on adding the message length bits to the pixels till the whole length is added and new pixels are formed. Then it simply replaces the originals pixels of the picture with the new encoded pixels. This is how the length of the message is encoded first.

Now to encode the message, the pixels after the encoded message length pixels are converted to binary and first two bits are neglected. Then again we take the bits of the pixels till the second last bit and add the bit of the message at the end. We continue this till the whole message is encoded and the new pixels are created. Now we just replace the original pixels with our new pixels.

Decoding:

In every cryptography mechanism ever made, decoding is always logically the exact opposite of the encoding process. The logic is hence not needed to be described all over again. Since, different members of the team wrote the code for the Client/Server, Encode and Decode functions/python files, the coding techniques employed may seem different in each function.

For decoding bitwise manipulation has been used to check the values stored in LSB in the two if-else conditions for checking length of message and content of message itself.

Since unlike the encoding, the decoding is being done at bit level and each relevant 1 and 0 is to be examined there are two conversion functions that go along with this. One is the built-in python “bin” function that takes integers and converts them into equivalent binary, the other is the binary to string function that is used after each go for a byte being stored in nine bits and being put into a char array. After the conversion there is a concatenation for each single character converted, so this could also be called a binary to character conversion since by a rule every time only a single character is stored in the array and converted.

Improvements to Page Load Time:

We have improved PLT for the browser page for encoding/decoding by ensuring in the code that unnecessary pixel values are not traversed and that once the length of the entire message has been stored and converted to a string we break out of the nested loop and proceed to displaying the message.

Another important PLT decreasing factor is that we used only one CSS file for styling, thus keeping the number of files to be loaded to a minimum.

All files that are to be loaded are also kept on server, and don’t have to be fetched from other internet resources, which further reduces PLT.

Limitations:

1. The length of the message depends on the pixels reserved for encoding the length. This is predefined in both encoding and decoding code.
2. The length of the message has to be small enough to be encoded in the picture
3. In encoding the message, we are wasting one bit for encoding each character. I.e. our code works such that 3 bits (RGB of each pixel) have to be encoded at once. Hence, the number of bits for encoding a character has to be a multiple of 3, whereas an ASCII of each character consists of 8 bits maximum.
4. The picture should be in RGB mode.
5. Images could have been compressed before sending and decompressed upon receiving in the client file.
6. An optional addition that ensures that only a person with a certain previously saved passcode (while encoding) has request the message since the entire point of hiding the message is for its existence to not be known. This is accomplished by sending an encrypted passcode file along with the image file and checking user enter password against the one saved in file. If both match the decoding goes forward, otherwise not. This could be a simple check but has not been implemented as the way to send the text file we would create right alongside the image file was not figured, as well as time was not spent on encryption/decryption of text.