Project Phase 2

Botz
Carl Yarwood, Garry Alcorn, Elisabeth Goggin

Overview

The goal of this project was to create a program used AES and steganography to hide a message in an image file. We decided to use specifically PNG image files due to the fact we found them easier to work with than other image file types. Though we found examples of more complex forms of steganography during our research in the earlier phase of this project, we chose to use a more straightforward method of steganography due to its ease of implementation. We decided that we did not wish to deal with the headache that came from messing with moving images, whether they be videos or a moving image file type like for example a GIF.

Interestingly, while working on our project, we found a lot of similarities between the methods we used and the methods used for corporate level watermarking. The method we used involved calculating the binary before anding it with a mask to encode the selected number of least significant bits.

The name of our program, BaldMan comes from our research in the earlier phase where we found references to an early story of steganography involving a slave getting the message tattooed on his head and then growing his hair out before being shaved for the message's delivery.

Challenges Faced

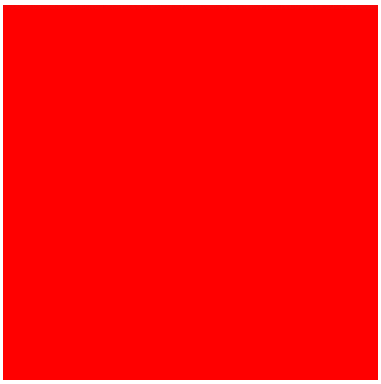
The largest challenge we faced when deciding out how to implement image steganography was figuring out what on the byte level of the images we could mess with. For example, avoiding control bits and such. We luckily managed to find a java class that would extract the image's info without touching the other stuff. Ultimately, we found it was much easier to put the message or image into the disguising image than to pull it out.

Test Cases

In the following example we will be hiding Figure 1a inside Figure 1b to get an output of Figure 1c.

(a) The image to be hidden.

(b) The image to hide the other image in.



(c) The output in which the image is hidden.

Figure 1: Example images

(a) This is an example of inserting too large of a message file into the image file causing distortion of the original image.

Discussion

For Bald Main I used the dream in code stegnogaphy post done by William Willard, for isnperation
and as an outline for the final code, we decided to use a terminal system fore ease of use used how to do it in java to help start AES. Also used various stack overflow sources.

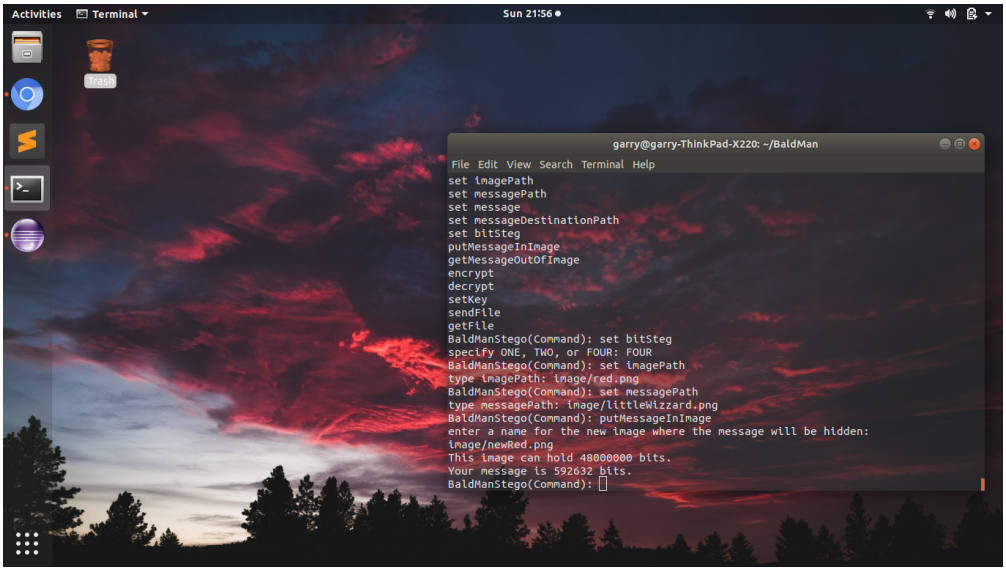


Figure 3: BaldMan running showing max bytes of the image.

Source Code

AESEncryption.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.security.spec.KeySpec;
import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;


public class AESEncryption {

   private static String theKey = "default";
   private static String salt = "E1F53135E559C253";

   //default Constructor
   public AESEncryption() {
   }
   //Return the current key being used for encryption/decryption
    public void getState(){
   System.out.println("Key => "+ theKey);
```

```
 }
//Put method to set the value for the key
public void setKey(String key) {
   theKey = key;
}

//Takes in a byte array and returns an encrypted byte array using the key,
    AES, SHA256 and PKCS5Padding
public static byte[] encrypt(byte[] byteToEncrypt ) {

   try {
      //This initializes the 16 byte of plain-text to be used in the 4x4 state
         matrix for AES to zero
      byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
      //This specifies the initialization vector that we use for DES in Cipher
         Block Chaining mode
      IvParameterSpec ivSpec = new IvParameterSpec(iv);
      //Password-Based Key Derivation function with Hash-based message
         Authentication code using SHA-256 as the secure hash Algorithm
      SecretKeyFactory factory =
         SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
      //Constructor that takes the password, the byte array of the salt, the
         iteration int, and the key length)
      KeySpec spec = new PBEKeySpec(theKey.toCharArray(), salt.getBytes(),
         65536, 256);
      //generates a secretKey object from the key specification
      SecretKey temp = factory.generateSecret(spec);
      //Constructs the secret key given the hashed key
      SecretKeySpec secretKey = new SecretKeySpec(temp.getEncoded(), "AES");
      //Creating a Cipher using the AES algorithm in CBC mode with PKCS5Padding
      Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
      //Initializes the cipher to encrypt the secret key to the Initialization
         Vector
      cipher.init(Cipher.ENCRYPT_MODE, secretKey, ivSpec);

      return cipher.doFinal(byteToEncrypt);
   }
   catch (Exception e) {
      e.printStackTrace();
   }
   return null;
}

//Takes in an encrypted byte array and Decrypts using AES CBC and PKCS5
    Padding, returning a decrypted byte array
public static byte[] decrypt(byte[] byteToDecrypt ) {
    try
```

```
    {
        byte[] iv = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
        IvParameterSpec ivspec = new IvParameterSpec(iv);

        SecretKeyFactory factory =
            SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
        KeySpec spec = new PBEKeySpec(theKey.toCharArray(), salt.getBytes(),
            65536, 256);
        SecretKey temp = factory.generateSecret(spec);
        SecretKeySpec secretKey = new SecretKeySpec(temp.getEncoded(), "AES");

        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, secretKey, ivspec);
        return cipher.doFinal(byteToDecrypt);
    }
    catch (Exception e) {
        System.out.println("Error while decrypting: " + e.toString());
    }
    return null;
}

//Converts the image and into a byte array using File IO
public byte[] getImageBytes(String imagePath) {
   byte[] content = null;
    File file = new File(imagePath);
       FileInputStream fis = null;
       try{
      fis = new FileInputStream(file);
      content = new byte[(int)file.length()];
      fis.read(content);
       }catch(FileNotFoundException e){
           System.out.println("File not found");
      return null;
       }catch(IOException e){
      System.out.println("Early IOException");
       }
       finally{
      try{
          if(fis != null){
         fis.close();
          }
      }
      catch(IOException e){
          System.out.println("IOException");
      }
       }
       return content;
```

```
   }
}
```

BaldMan.java

```
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.FileNotFoundException;
import java.io.ByteArrayInputStream;
import java.io.FileOutputStream;
import java.awt.Point;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.awt.image.WritableRaster;
import java.awt.image.Raster;
import java.awt.image.DataBufferByte;
import java.nio.ByteBuffer;

import javax.imageio.ImageIO;



/**
BaldMan is a Stegnography class used for hiding files or messages in png images,
it is meant to be easily implemented in a commandline or gui enviorment
 */
public class BaldMan{

    private String imagePath = null;
    private String message = null;
    private String messagePath = null;
    private String messageDestinationPath = null;
    private Bits bitSteg = Bits.ONE;

    public BaldMan(){
    }
    /**
     *Prints the current state of the program to the console
     */
    public void getState(){
   System.out.println("imagePath => " + imagePath);
   System.out.println("message => " + message);
   System.out.println("messagePath => " + messagePath);
   System.out.println("messageDestinationPath => " + messageDestinationPath);
   System.out.println("bitSteg => " + bitSteg);
    }
```

```
 /**
  *Uses the current state of the program to try to put a message in an image
  *@param String representing path to new image
 */

 public void putMessageInImage(String newImageName){
if(imagePath == null){
    System.out.println("Must set imagePath");
    return;
}
else if ( message == null && messagePath == null){
    System.out.println("must set message or message Path");
    return;
}
byte[] encodeMessage = getMessage();
if(encodeMessage == null){
    System.out.println("Message Not found");
    return;
}
BufferedImage img = getImageCopy(getImage());
byte [] image = convertImage(img);
if(bitSteg == Bits.ONE){
    System.out.println("This image can hold " + image.length + " bits.");
}
else if(bitSteg == Bits.TWO){
    System.out.println("This image can hold " + image.length * 2 + " bits.");
}
else if(bitSteg == Bits.FOUR){
    System.out.println("This image can hold " + image.length * 4 + " bits.");
}
System.out.println("Your message is "+ encodeMessage.length * 8 + " bits.");
if(image == null){
    System.out.println("could not get image");
    return;
}
try{
    encodeMessage(encodeMessage, image);
}catch(IOException e){
    System.out.println("Message too large for image");
    return;
}
try{
    ImageIO.write(img,"png",new File(newImageName));
}catch(Exception e){
    System.out.println("could not write file");
    return;
}
```

```
 }


 /**
  *uses the current state of the program to
  *try and pull an image form a message
 */

 public void getMessageOutOfImage(){
byte letIn = 0;
byte divisor = 1;
if(imagePath == null){
    System.out.println("Must set an Image Path");
}
if(bitSteg == Bits.ONE){
    divisor = 1;
    letIn = 1;
}
else if (bitSteg == Bits.TWO){
    divisor = 2;
    letIn = 3;
}
else if (bitSteg == Bits.FOUR){
    divisor = 4;
    letIn = 15;
}
byte[] img = convertImage(getImage());
int length = 0;
int posInImage = 0;
for(int i = 0; i<(32/divisor); i++){
    length =length << divisor;
    length = length | img[i] & letIn;
    posInImage++;
}
byte[] msg = new byte[length/(8/divisor)];
for(int i = 0; i < msg.length; i++){
    for(int c = 0; c< (8/divisor); c++){
   msg[i] = (byte)(msg[i] << divisor);
   msg[i] = (byte)(msg[i]| (img[posInImage]& letIn));
   posInImage++;
    }
}
if(messageDestinationPath == null){
         String message = new String(msg);
    System.out.println(message);
```

```
}
else{
    try(FileOutputStream fos = new FileOutputStream(messageDestinationPath)){
   fos.write(msg);
    }catch(Exception e){
   System.out.println("cannot write message to destination");
    }
}
 }

 /**
  *Sets imagePath in state of program
  *@param string representing path to image
  */
 public void setImagePath(String imagePath){
this.imagePath = imagePath;
 }
 /**
  *Sets the path to the ouput file for message
  *@param string representing message output
  */

 public void setMessageDestinationPath(String Path){
this.messageDestinationPath = Path;
 }
 /**
  *sets path to message
  *@param string represent path to file holding message
  */
 public void setMessagePath(String messagePath){
this.messagePath = messagePath;
this.message = null;
 }
 /**
  *sets Message
  *@param String representing message
  */
 public void setMessage(String message){
this.message = message;
this.messagePath = null;
 }
 /**
  *set type of stnography, leastSignificant, two least, four least
  *@parm Bit enum which represents type of encryption
  */
 public void setStegBits( Bits b){
this.bitSteg = b;
```

```
 }
 /**
  *gets message based on object state
  *@return byte[] for message
  */
 private byte[] getMessage(){
byte[] content = null;
if(messagePath == null && message == null){
    System.out.println("need to set message or messagePath");
    return null;
}
else if( messagePath == null){
    return message.getBytes();
}
else{
    File file = new File(messagePath);
    FileInputStream fis = null;
    try{
   fis = new FileInputStream(file);
   content = new byte[(int)file.length()];
   fis.read(content);
    }catch(FileNotFoundException e){
        System.out.println("File not found");
   return null;
    }catch(IOException e){
   System.out.println("Early IOException");
    }
    finally{
   try{
      if(fis != null){
     fis.close();
      }
   }
   catch(IOException e){
      System.out.println("IOException");
   }
    }
    return content;
}
 }
 /**
  *gets copy of buffer image
  *@param image to be coppied
  *@return copy of image
  */
 private BufferedImage getImageCopy(BufferedImage image){
BufferedImage imageCopy = new BufferedImage(image.getWidth(),
```

```
    image.getHeight(), BufferedImage.TYPE_3BYTE_BGR);
Graphics2D draw = imageCopy.createGraphics();
draw.drawRenderedImage(image,null);
draw.dispose();
return imageCopy;
 }
 /**
  *gets buffered image base on program state
  *@return image
  */
 private BufferedImage getImage(){
BufferedImage img = null;
try{
    File imageFile = new File(imagePath);
    img = ImageIO.read(imageFile);
}catch(FileNotFoundException e){
    System.out.println("Image File Not Found");
    return null;
}catch(IOException e){
    System.out.println("Error on Read Try New Image");
    return null;
}
return img;
 }
 /**
  *converts int ot byte array
  *@param int to be converted
  *@return byte[] array for int
  */
 private byte[] convertInt(int i){
ByteBuffer buff = ByteBuffer.allocate(4);
buff.putInt(i);
return buff.array();
 }
 /**
  *Converts a byte array to an int
  *@param byte array to be converted
  *@return int
  */
 private int convertBackInt(byte[] num){
ByteBuffer buff = ByteBuffer.wrap(num);
return buff.getInt();
 }
 /**
  *converts bufferd image to byte array representing pixels with Raster
  *@param image to be converted
  *@return byte[] representing image
```

```
  */
 private byte[] convertImage(BufferedImage img){
    Raster raster = (Raster)img.getRaster();
DataBufferByte buffer = (DataBufferByte) raster.getDataBuffer();
return buffer.getData();
 }
 /**
  *does the heavy lifting of putting the message into the image
  *@param byte array representing message
  *@param byte array representing image
  */
 private void encodeMessage(byte[] message, byte[] img)throws IOException{
byte mask = 0;
byte letIn = 0;
int divisor = 1;
if(bitSteg == Bits.ONE){
    mask = (byte)254;
    letIn = 1;
    divisor = 1;
}
else if(bitSteg == Bits.TWO){
    mask = (byte)252;
    letIn = 3;
    divisor = 2;
}
else if (bitSteg == Bits.FOUR){
    mask = (byte)240;
    letIn = 15;
    divisor = 4;
}
if(img.length < (message.length * (8/divisor) + (32/divisor))){
    System.out.println("Message too large for given image");
    throw new IOException("image not big enough for message");
}
byte[] messageLength = convertInt(message.length * (8/divisor));
int currentPosInImage = 0;
for(int i = 0; i< 4; i++){
    for(int bits = (8/divisor) - 1; bits >= 0 ; bits--){
   img[currentPosInImage] =(byte)(img[currentPosInImage] & mask);
   img[currentPosInImage] =(byte) (img[currentPosInImage] | ((messageLength[i]
      >> (bits * divisor)) & letIn ));
   currentPosInImage ++;
    }
}
for(int i = 0; i< message.length; i++){
    for(int bits = (8/divisor) - 1 ; bits >= 0 ; bits--){
   img[currentPosInImage] =(byte)(img[currentPosInImage] & mask);
```

```
      img[currentPosInImage] =(byte) (img[currentPosInImage] | ((message[i] >>
         (bits*divisor)) & letIn ));
      currentPosInImage ++;
       }
   }
    }
}
```

Bits.java

```
enum Bits{
    ONE, TWO, FOUR;
}
```

Main.java

```
import java.util.Scanner;
import java.io.FileOutputStream;
import java.io.File;
import java.io.FileInputStream;
public class Main{

    public static void main(String[] args){
   boolean keepGoing = true;
   BaldMan stego = new BaldMan();
   AESEncryption aes = new AESEncryption();
   Scanner scan = new Scanner(System.in);
   String command = "";
   String CommandLineState = "BaldManStego(Command): ";
   while(keepGoing){
       System.out.print(CommandLineState);
       command = scan.nextLine();
       if(command.equals("state")){
      stego.getState();
      aes.getState();
       }
       else if(command.equals("quit") || command.equals("q")){
      keepGoing = false;
       }
       else if(command.equals("set imagePath")){
      System.out.print("type imagePath: ");
      String imagePath = scan.nextLine();
      stego.setImagePath(imagePath);
       }
       else if(command.equals("set messagePath")){
          System.out.print("type messagePath: ");
          String messagePath = scan.nextLine();
```

```
    stego.setMessagePath(messagePath);
 }
 else if(command.equals("set message")){
System.out.print("type message: ");
String message = scan.nextLine();
stego.setMessage(message);
 }
 else if(command.equals("set messageDestinationPath")){
System.out.print("type messageDestinagionPath: ");
String MessageDestination = scan.nextLine();
stego.setMessageDestinationPath(MessageDestination);
 }
 else if(command.equals("set bitSteg")){
System.out.print("specify ONE, TWO, or FOUR: ");
String newBitSteg = scan.nextLine();
if(newBitSteg.equals("ONE")){
    stego.setStegBits(Bits.ONE);
}else if(newBitSteg.equals("TWO")){
    stego.setStegBits(Bits.TWO);
}else if(newBitSteg.equals("FOUR")){
    stego.setStegBits(Bits.FOUR);
}else{
    System.out.println("no valid entry made");
}
 }
 else if(command.equals("putMessageInImage")){
System.out.print("enter a name for the new image where the message will be
    hidden: ");
String newImgFileName = scan.nextLine();
stego.putMessageInImage(newImgFileName);
 }
 else if(command.equals("getMessageOutOfImage")){
stego.getMessageOutOfImage();
 }
 else if(command.equals("encrypt")){
System.out.print("enter file you would like encrypt: ");
String encryptFile = scan.nextLine();
System.out.print("enter the name of the new encrypted file: ");
String newFileName = scan.nextLine();
byte[] content = aes.encrypt(aes.getImageBytes(encryptFile));
try(FileOutputStream fos = new FileOutputStream(newFileName)){
    fos.write(content);
}catch(Exception e){
    System.out.println("could not write file");
}
 }
 else if(command.equals("decrypt")){
```

```
System.out.print("enter the name of the file you would like to decrypt: ");
String decryptFile = scan.nextLine();
System.out.print("enter the name of the file you would like to store the
   new decrypted file in: ");
String newFile = scan.nextLine();
byte[] content = aes.decrypt(aes.getImageBytes(decryptFile));
try(FileOutputStream fos = new FileOutputStream(newFile)){
   fos.write(content);
}catch(Exception e){
   System.out.println("could not write file");
}
 }
 else if(command.equals("set key")){
System.out.print("Type what you would like your key to be");
String key = scan.nextLine();
aes.setKey(key);
 }
 else if(command.equals("sendFile")){
System.out.print("Enter the number of files you would like to send: ");
int numFiles = scan.nextInt();
String dummy = scan.nextLine();
System.out.print("Enter the port you wold like to use: ");
int port = scan.nextInt();
dummy = scan.nextLine();
String[] fileNames = new String[numFiles];
for(int i = 0; i < numFiles; i++){
   System.out.print("Enter the name of file " + i + ": ");
   fileNames[i] = scan.nextLine();
}
TCPServerFile fs = new TCPServerFile();
fs.createSocket(port);
for(int i = 0; i < numFiles; i++){
   fs.sendFile(fileNames[i]);
}
fs.shutDown();
 }
 else if(command.equals("getFile")){
System.out.print("Enter the number of Files you would like to receive: ");
int numFile = scan.nextInt();
String dummy = scan.nextLine();
System.out.print("Enter the ip address you want to receive form: ");
String ip = scan.nextLine();
System.out.print("Enter the port of connection: ");
int port = scan.nextInt();
dummy = scan.nextLine();
String[] newFileName = new String[numFile];
for(int i = 0; i < numFile; i++){
```

```
    System.out.print("Enter the name of recived File " + i +": ");
    newFileName[i] = scan.nextLine();
}
TCPClientFile soc = new TCPClientFile();
soc.createSocket(ip, port);
for(int i = 0; i< numFile; i++){
    soc.receiveFile(newFileName[i]);
}
soc.shutDown();
 }
 else if(command.equals("readKeyFromFile")){
System.out.print("Enter path to file containing key: ");
String keyFileName = scan.nextLine();
File file = new File(keyFileName);
FileInputStream fis = null;
byte[] content = null;
try{
    fis = new FileInputStream(file);
    content = new byte[(int) file.length()];
    fis.read(content);
}catch(Exception e){
    System.out.println("Could not read file");
}finally{
    try{
   if(fis != null){
      fis.close();
   }
    }catch(Exception e){
   System.out.println("Issue closing File");
    }
}
String key = new String(content);
aes.setKey(key);
 }
 else if(command.equals("-h")){
System.out.println("state (Recomended)");
System.out.println("quit");
System.out.println("set imagePath");
System.out.println("set messagePath");
System.out.println("set message");
System.out.println("set messageDestinationPath");
System.out.println("set bitSteg");
System.out.println("set key");
System.out.println("putMessageInImage");
System.out.println("getMessageOutOfImage");
System.out.println("encrypt");
System.out.println("decrypt");
```

```
      System.out.println("sendFile");
      System.out.println("getFile");
      System.out.println("readKeyFromFile");
       }
       else{
      System.out.println("Use -h to get a list of valid commands");
       }

   }
    }

}
```

Main.java

```
import java.io.*;
import java.net.Socket;
import java.net.SocketException;
import java.net.UnknownHostException;

/**
 *
 * @author cjaiswal
8
 *
 * UPDATED: 12/2/2018 by Botz
 *
 * A simple TCP client for file download with an expected typical tcp payload
     size of 1000 bytes
 *
 */

public class TCPClientFile
{
    private Socket socket = null;
    private DataInputStream inStream = null;
    private DataOutputStream outStream = null;

    public TCPClientFile()
    {

    }

    public void createSocket(String ip, int port)
    {
        try
        {
```

```
      //connect to localHost (or ip) at given port #
         socket = new Socket(ip , port);
         System.out.println("Connected");
         //fetch the streams
         inStream = new DataInputStream(socket.getInputStream());
         outStream = new DataOutputStream(socket.getOutputStream());
      }
      catch (Exception u)
      {
         u.printStackTrace();
      }
   }

   public void receiveFile(String newFileName)
   {
     byte [] data = null;
     //decide the max buffer size in bytes
     //a typical value for a tcp payload is 1000 bytes, this is because of
     //the common MTU of the underlying ethernet of 1500 bytes
     //HOWEVER their is no optimal value for tcp payload, just a best guess i.e.
        1000 bytes
     final int MAX_BUFFER = 1000;
     try
     {
        //read the size of the file <- coming from Server
        long fileSize = inStream.readLong();
        int bufferSize=0;

        //decide the data reading bufferSize
        if(fileSize > MAX_BUFFER)
           bufferSize = MAX_BUFFER;
        else
           bufferSize = (int)fileSize;

        data = new byte[bufferSize];

        //insert the path/name of your target file
        FileOutputStream fileOut = new FileOutputStream(newFileName,true);

        //now read the file coming from Server & save it onto disk

        long totalBytesRead = 0;
        while(true)
        {
           //read bufferSize number of bytes from Server
           int readBytes = inStream.read(data,0,bufferSize);
```

```
            byte[] arrayBytes = new byte[readBytes];
            System.arraycopy(data, 0, arrayBytes, 0, readBytes);
            totalBytesRead = totalBytesRead + readBytes;

            if(readBytes>0)
            {
               //write the data to the file
               fileOut.write(arrayBytes);
               fileOut.flush();
            }

            //stop if fileSize number of bytes are read
            if(totalBytesRead == fileSize)
               break;

            //update fileSize for the last remaining block of data
            if((fileSize-totalBytesRead) < MAX_BUFFER)
               bufferSize = (int) (fileSize-totalBytesRead);

            //reinitialize the data buffer
            data = new byte[bufferSize];
         }
         System.out.println("File Size is: "+fileSize + ", number of bytes read
             are: " + totalBytesRead);

         //socket.close();
         fileOut.close();
         //inStream.close();
      }
      catch(Exception e)
      {
         e.printStackTrace();
      }
    }
    public void shutDown(){
   try{
       socket.close();
       inStream.close();
   }catch(Exception e){
       e.printStackTrace();
   }
    }
}
```

Main.java

```
import java.io.*;
```

```java
import java.io.ObjectOutputStream.PutField;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.SocketException;

/**
 *
 * @author cjaiswal
 *
 * UPDATED: 12/2/2018 by botz
 *
 * A simple TCP server for file download with a typical tcp payload size of 1000
    bytes
 *
 */

public class TCPServerFile
{
    private ServerSocket serverSocket = null;
    private Socket socket = null;
    private DataInputStream inStream = null;
    private DataOutputStream outStream = null;

    public TCPServerFile()
    {

    }

    public void createSocket(int port)
    {
        try
        {
         //create Server and start listening
            serverSocket = new ServerSocket(port);
            //accept the connection
            socket = serverSocket.accept();
            //fetch the streams
            inStream = new DataInputStream(socket.getInputStream());
            outStream = new DataOutputStream(socket.getOutputStream());
            System.out.println("Connected");
        }
        catch (IOException io)
        {
            io.printStackTrace();
        }
    }
```

```
public void sendFile(String FileName)
{
  final int MAX_BUFFER = 1000;
  byte [] data = null;
  int bufferSize = 0;
  try
  {
     // write the filename, relative path, below in the File constructor
     File file = new File(FileName);
     FileInputStream fileInput = new FileInputStream(file);
     //get the file length
     long fileSize = file.length();

     System.out.println("File size at server is: " + fileSize + " bytes");
     //first send the size of the file to the client
     outStream.writeLong(fileSize);
     outStream.flush();

     //Now send the file contents
     if(fileSize > MAX_BUFFER)
        bufferSize = MAX_BUFFER;
     else
        bufferSize = (int)fileSize;

     data = new byte[bufferSize];

     long totalBytesRead = 0;
     while(true)
     {
        //read upto MAX_BUFFER number of bytes from file
        int readBytes = fileInput.read(data);
        //send readBytes number of bytes to the client
        outStream.write(data);
        outStream.flush();

        //stop if EOF
        if(readBytes == -1)//EOF
           break;

        totalBytesRead = totalBytesRead + readBytes;

        //stop if fileLength number of bytes are read
        if(totalBytesRead == fileSize)
           break;

        ////update fileSize for the last remaining block of data
        if((fileSize-totalBytesRead) < MAX_BUFFER)
```

```
            bufferSize = (int) (fileSize-totalBytesRead);

          //reinitialize the data buffer
          data = new byte[bufferSize];
        }

        fileInput.close();
        //serverSocket.close();
        //socket.close();
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
    }
    public void shutDown(){
   try{
       serverSocket.close();
       socket.close();
   }catch(Exception e){
       e.printStackTrace();
   }
    }

}
```

Project Contributions

Research/Brainstorming	Carl Yarwood
	Garry Alcorn
	Elisabeth Goggin
Coding	Carl Yarwood
	Garry Alcorn
Writeup	Elisabeth Goggin
	CarlYarwood
	GarrayAlcorn