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

Discussion

Max Bytes

Source Code

AESEncryption.java

```
import java.io.UnsupportedEncodingException;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.spec.SecretKeySpec;

public class AESEncryption {

   private static SecretKeySpec secretKey;
   private static byte[] key;

   public static void createKey(String myKey) {
      MessageDigest sha = null;
      try {
         key = myKey.getBytes("UTF-8");
         sha = MessageDigest.getInstance("SHA-256");
         key = sha.digest(key);
         key = Arrays.copyOf(key, 16);
         secretKey = new SecretKeySpec(key, "AES");
      }
      catch (NoSuchAlgorithmException e) {
         e.printStackTrace();
      }
      catch (UnsupportedEncodingException e) {
         e.printStackTrace();
      }
   }

   public static String encrypt(String strToEncrypt, String secret) {
      try {
         createKey(secret);
         Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
```

```
            cipher.init(Cipher.ENCRYPT_MODE, secretKey);
            return
                Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));
        }
        catch (Exception e) {
            e.printStackTrace();
        }
        return "failed";
    }

    public static String decrypt(String strToDecrypt, String secret) {
        try {
            createKey(secret);
                Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
                cipher.init(Cipher.DECRYPT_MODE, secretKey);
                return new
                    String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));
            }
            catch (Exception e)
            {
                System.out.println("Error while decrypting: " + e.toString());
            }
            return null;
        }


    public static void main(String[] args) {
        final String secretKey = "My secret key";

        String originalString = "TEST";
        String encryptedString = AESEncryption.encrypt(originalString, secretKey) ;
        String decryptedString = AESEncryption.decrypt(encryptedString, secretKey)
            ;

        System.out.println(originalString);
        System.out.println(encryptedString);
        System.out.println(decryptedString);
    }

}
```

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;



//if have time implement this as a runnable for multi threading
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(){
    }

    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;
}


 }



 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{
    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");
    }
}
 }


 public void setImagePath(String imagePath){
this.imagePath = imagePath;
 }

 public void setMessageDestinationPath(String Path){
this.messageDestinationPath = Path;
 }



 public void setMessagePath(String messagePath){
this.messagePath = messagePath;
this.message = null;
 }



 public void setMessage(String message){
this.message = message;
```

```
this.messagePath = null;
 }

 public void setStegBits( Bits b){
this.bitSteg = b;
 }




 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;
}
 }
```

```
 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;
 }


 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;
 }

 //takes and int and convertes into a byta array by shifting the int
 // and anding it with the byte 11111111, as 1 and 1 = 1, and 1 and 0 = 0
 private byte[] convertInt(int i){
ByteBuffer buff = ByteBuffer.allocate(4);
buff.putInt(i);
return buff.array();
 }



 private int convertBackInt(byte[] num){
ByteBuffer buff = ByteBuffer.wrap(num);
return buff.getInt();
 }
 private byte[] convertImage(BufferedImage img){
     Raster raster = (Raster)img.getRaster();
DataBufferByte buffer = (DataBufferByte) raster.getDataBuffer();
return buffer.getData();
 }
 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{
        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;
public class Main{

    public static void main(String[] args){
   //Scanner scan = new Scanner(System.in);
   // System.out.print("Enter the port you would like your server on: ");
   // int port = scan.nextInt();
   //TCPServer server = new TCPServer(port);
   //server.createSocket();
   // while(true){
   //   server.listenForConnection();
   // }
   BaldMan stego = new BaldMan();
   stego.setImagePath("red.png");
        stego.setMessagePath("littleWizzard.png");
   stego.setMessageDestinationPath("outLittleWizzard.png");
   stego.setStegBits(Bits.FOUR);
   stego.putMessageInImage("secretRed.png");
   stego.setImagePath("secretRed.png");
   stego.getMessageOutOfImage();
    }

}
```

Project Contributions

Research/Brainstorming	Carl Yarwood
	Garry Alcorn
	Elisabeth Goggin
Writeup	Elisabeth Goggin