Steganography Using Computer Images

William Barratt

June 13, 2003

Abstract

This project investigates a method for encoding text messages in computer image files, a process called steganography. Steganography is not well-known to the public but has existed in different forms for thousands of years. The usefulness of steganography is not confined to sending secret messages; it is also possible to use hidden data to efficiently store an accompanying caption with an image. Through judicious choice of file type and algorithm, it is possible to hide data in an image with no effect detectable by methods other than pixel-by-pixel comparison with a known original. Because of the vast number of image files (including both photographs and design elements that are frequently used in Web pages) available on the Internet, steganography can go undetected if carefully created and placed.

1 Introduction

Steganography is a term for processes that hide information within some medium. For exam-

ple, the microdot images used by spies during the Cold War were a kind of steganography. In recent years, steganography with computers has grown in popularity because of the ability to more easily place steganographic data within images, audio files, etc. This project's goal is to develop and investigate methods for hiding text messages in computer image files.

2 Background

The idea of hiding data within computer images is not new. Multiple programs already exist for this purpose, and some are available for download on the Internet. These programs, such as JPHIDE[2] and GZSteg[5] take images in common file formats like JPEG and modify them slightly to include a user-specified text message. Each has its limitations, and other programs have been created to detect images that have been processed by the steganography programs[3]. For example, steganography with JPEG images is more noticeable, due to the lossy compression algorithm JPEG uses. In the compression, the raw image is changed and muddied significantly, and the difference between the original and the modified image is magnified. The change may also effect a difference inthe disk space each image occupies. The algorithm chosen also affects the detectability of the steganography. If an algorithm that hides large amounts of information is used, there will be a much greater change in the image's appearance. Therein lies the challenge and trade-off of steganography: hide the most data with the least noticeable (to both humans and computers) change in the image.

3 Materials

The project was coded and tested on AMD Athlon-based and dual Intel Celeron-based computers running Debian Linux. All programs were written in C++ and compiled using the standard GNU C++ compiler "g++".

4 Procedure

First, an algorithm for the steganography must be selected. For simplicity, an algorithm that changed only the lowest-order bit of each pixel color value was selected. Next, the algorithm must be implemented on one or more types of image file. The first file type chosen was the P2 PGM standard. P2 PGM images consist of an array of numbers in ASCII text, each representing a grayscale pixel value from 0 to 255. This attribute of the P2 PGM makes it ideally suited for a simple steganography program because of the ease with which it source data can be read not only by a computer but also by a human.

Figure 1. fig1.pgm

This first iteration of the program reads in a PGM P2 image file, named fig1.pgm. After analyzing the file's headers, it reads the pixel values into an integer array. The program then asks the user for the message to be hidden in the image. While it's possible to store any data, this iteration is limited to ASCII text. The program then makes each low-order bit of each pixel value match the sequence of bits that are equivalent to the text string the user input. This method is simple, but it is effective; no pixel changes color by more than 1 (out of 256 shades of gray), some do not change at all, and the output file size is identical to that of the original[1]. The limitation of this algorithm is that the maximum number of characters that can be hidden is the number of pixels divided by seven (because there are seven bits in an ASCII character).

5 Results

5.1 Hiding the data

The result of the first iteration is a UNIX command-line program. The command-line parameters are the input file and output file; if one of these is missing, the program prompts the user for these data. Upon successful opening of the input file, the user is given the maximum number of characters allowed and asked to input a message to hide in the output image. After changing the necessary pixel values, the output image file is written. The input and output files are identical in size and the minute change in appearance is not noticeable

unless the actual pixel values are compared.

Figure 2. message.pgm. Note the lack of perceptible difference between this image and fig1.pgm (Figure 1).

5.2 Retrieving the data

Now a second command-line program is used to retrieve the data from the modified image. The command-line argument is the filename of the modified file. A common misconception is that one must have access to the original image in order to extract the data, but this is not the case. The "desteganography" program simply reads all the low-order bits of the binary representations of the pixel values as ASCII characters. If the program reads in ASCII character 7 (the "beep" character), it knows that it has reached the end of the message.

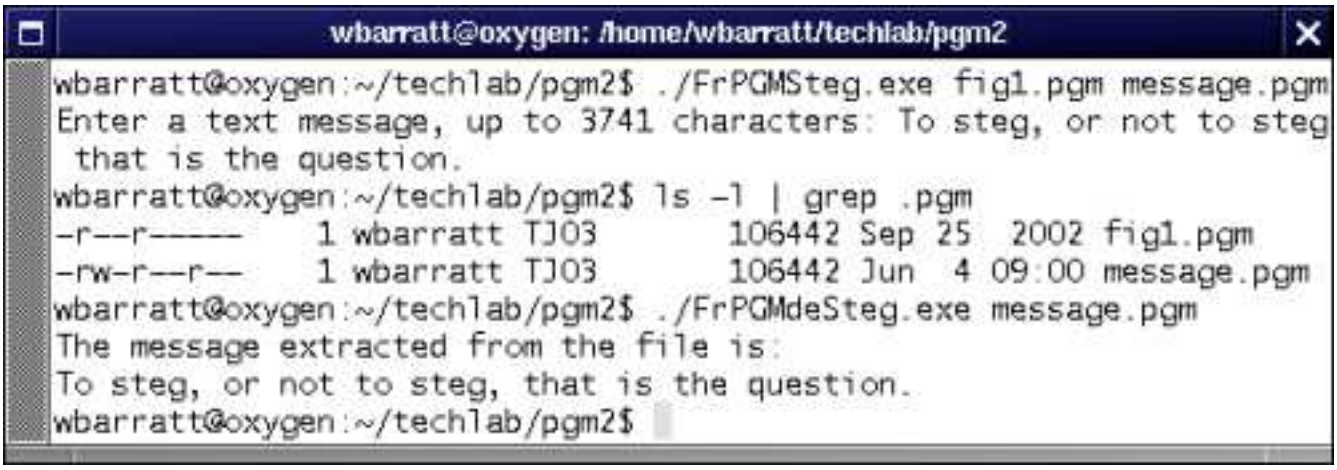

Figure 3. An example run of the program in the terminal.

6 Discussion

6.1 Success

The success of the steganography algorithm is twofold: no change in the image's size on disk, and no change in appearance. The only way to detect that the image has been modified is by using a computer to compare it to an image that is *positively known* to be the original. Because of the uncertainty of the reason for the differences between the two images, it may not be realized that the steganographic data is there. For example, if the original image were converted to another file type and then converted back, some changes in the pixel values might occur. This leaves reasonable doubt to an outsider that the changes may not indicate steganography.

6.2 Problems

Unfortunately, this particular implementation has problems. The algorithm used is very simple, making it very easy for a suspicious person to extract the hidden data. An algorithm that includes encryption would make it more difficult for the data to be extracted and decoded should the steganography be discovered.

The second shortcoming is the relatively small amount of data that can be stored using this file type and algorithm. By using only the low-order bits to store data, the amount of data that can be stored (in bytes) is the number of pixels divided by eight. In the example image (see Figure 1), which is 131 pixels wide and 200 pixels high, there is space for only about 3 kilobytes of data, or about 3700 7-bit ASCII characters. By using the two lowest-order bits, more data could be stored, but the change might become noticeable to the human eye.

Another problem is the lack of usefulness of the PGM format. It is an uncompressed image file type, and is very uncommonly used on the Internet and World Wide Web. A successful steganography program needs to use a common, unsuspicious file type such as JPEG, GIF, or PNG. For further development of this project, PNG is an excellent choice because it is not only lossless, but it has open-source code available for developers working with PNG[4].

7 Conclusion

This project was a success overall. A simple but effective steganography algorithm was designed and implemented using P2 PGM files. The algorithm was able to hide the data without changing the file size at all, and without changing the image's appearance significantly. As is, the project can be useful for some; the PGM files can be converted to PNG and converted back without losing the hidden data. Although the project is simple, the command-line nature of the program and the fact that it has only been tested in Linux make it unready for the mass market, especially compared to more venerable products like JPHIDE.

The future of this project is uncertain, but there are several areas for expansion. First is the possibility of expansion into different file types, such as PNG or even JPEG. These would both take significant amounts of time to program, and it may not be possible to devote much effort to expanding the program's capabilities. The second possibility is the creation and use of a better algorithm. This option would be less time-consuming, because most of the program's structure could remain unchanged. The third option is development into a graphical user interface. This is the least likely to happen, due to the sharp learning curve and very time-consuming nature of such an undertaking, for only a small increase in usefulness.

References

[1] Kay, Russell. How Steganography Works URL

http://www.computerworld.com/securitytopics/security/story/0,10801,71728,00.html

(visited 2003, May 20).

[2] Latham, Allan. Steganography URL http://linux01.gwdg.de/ alatham/stego.html

(visited 2002, October 16).

[3] Provos, Niels. Steganography Detection with Stegdetect URL

http://www.outguess.org/detection.php (visited 2002, October 23).

[4] Roelofs, Greg. Libpng Home URL

http://www.libpng.org/pub/png/libpng.html (visited 2003, June 11).

[5] Wilson, Preston. Linkbeat.com URL

http://www.linkbeat.com/files (visited 2002, October 16).

8 Appendix A: Code

File 1, stegFr.cpp, is a steganography program for P2 or P3 PGM images.

```
//Copyright 2002 William Barratt

//stegFr.cpp

/* User-friendly steganography program for .pgm files.  */
```

```
#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void handleArgs(int argc, char* argv[],
        FILE *& infile, FILE *& outfile);
void readWriteHeaders(FILE * infile, FILE * outfile,
        int & h, int & w, int & max);
void readIntoVector(FILE * infile, int* vector);
bool getMessage(char * input, int h, int w);
void modifyPixels(int* vector, char * input);
void writeNewPixels(FILE * outfile, int* vector, int h, int w);

int main(int argc, char* argv[])
{
  FILE * infile;
  FILE * outfile;

  handleArgs(argc, argv, infile, outfile);  //get filenames
```

```
int h, w, max;                      //image height, width,
                                    //and max. pixel value

//read headers of source file and write headers of new file
readWriteHeaders(infile, outfile, h, w, max);

int* vector = new int[h*w];         //vector for pixel values

readIntoVector(infile, vector);     //read pixels into vector

fclose(infile);                     //close source file

char * input = new char[h*w/7];     //string for message
if(!getMessage(input, h, w))        //get message, error test
{
  fclose(outfile);
  system("rm message.pgm");
  exit(2);                         //return code 2
}

modifyPixels(vector, input);        //put message in pixels
```

```
  writeNewPixels(outfile, vector, h, w); //write pixels to file


  fclose(outfile);                    //close output file


  return 0;
}


void handleArgs(int argc, char* argv[],
        FILE *& infile, FILE *& outfile)
{
  char* file = new char[80];          //temporary string


  if(argc > 3)                        //if 3 filenames entered
  {
    cout << "Too many command-line arguments; aborting."
         << endl;
    exit(3);
  }
  if(argc == 2)                       //if 1 filename entered
  {
    infile = fopen(argv[1], "r");
```

```
    cout << "Name of output image file: ";
    cin >> file;
    outfile = fopen(file, "w");
  }
  else if(argc == 3)                //if 2 filenames entered
  {
    infile = fopen(argv[1], "r");
    outfile = fopen(argv[2], "w");
  }
  else if(argc == 1)                //if no filenames entered
  {
    cout << "Name of input image file: ";
    cin >> file;
    infile = fopen(file, "r");

    cout << "Name of output image file: ";
    cin >> file;
    outfile = fopen(file, "w");
  }
}
void readWriteHeaders(FILE * infile, FILE * outfile,
```

```
        int &h, int &w, int & max)
{
  char* str = new char[80];          //string for reading

  fprintf(outfile,"%s\n", "P2");     //write initial filetype header

  fgets(str, 80, infile);            //reads first line

  if(str[0] != 'P' || str[1] != '2') //check header
  {
    fclose(infile);
    fclose(outfile);
    system("rm message.pgm");
cout << "File header indicates wrong file type" << endl
    << "File must be .pgm with 'P2' in first line" << endl;

exit(1);                          //end with exit code 1
  }

  fgets(str, 80, infile);            //second line (place for comments)
  while(str[0] == '#')               //detects comment line
```

```
  {
    fprintf(outfile, "%s", str);     //copy comment line
    fgets(str, 80, infile);          //read next line
  }

    //retrieval and copying of width and height
  char* width = new char[80];
  char* height = new char[80];

  int spaceLoc = strcspn(str, " ");  //finds space between w & h
  for(int x = 0; x < spaceLoc; x++)
  {
width[x] = str[x];               //copies width from temp string
  }
  w = atoi(width);                   //convert to int

  height = strstr(str, " ");         //get substring for height
  h = atoi(height);                  //convert to int

  fprintf(outfile, "%d %d\n", w, h);
```

```
  fgets(str, 80, infile);            //read final header
  max = atoi(str);                   //copy to max for later use
  fprintf(outfile, "%d\n", max);     //write max to new file
}


void readIntoVector(FILE *infile, int* vector)
{
  char* line = new char[80];         //buffer for reading lines
  char* tmpstr = new char[80];       //temp for manipulation

  int place = 0;                     //vector index

  while(!feof(infile))               //while not at file end
  {
    fgets(line, 80, infile);         //get one line
    tmpstr = strtok(line, " ");      //gets pixel value

    while(tmpstr != NULL)            //while not at line end
    {
      if(strcspn(tmpstr, "0123456789") != strlen(tmpstr))
      {                              //checks for numbers in substring
```

```
        vector[place] = atoi(tmpstr);//places number in vector
        place++;                     //increments index
      }
      tmpstr = strtok(NULL, " ");    //find next value
    }
  }
}

bool getMessage(char * input, int h, int w)
{
  //get text message input from user

  cout << "Enter a text message, up to "<<(h*w/7)-1<<" characters: ";
  cin.getline(input, h*w);

  //check to make sure the message is not too long
  if(strlen(input) > ((h * w / 7) - 1))
  {
    cout<<"ERROR: Message is too long. Aborting program.";
    return false;
  }
```

```
  else return true;
}


void modifyPixels(int* vector, char * input)
{
  char * tmpstr = new char[1];       //temporary string
  tmpstr[0] = char(7);               //beep char. denotes msg end
  strcat(input, tmpstr);             //add message end character

  int length = strlen(input), strIndex, shiftNum, vectorIndex = 0;

  for(strIndex = 0; strIndex < length; strIndex++)
  {
    for(shiftNum = 6; shiftNum >= 0; shiftNum--)
    {
      //bitwise ops. mask set low order bit to desired value
      vector[vectorIndex] = (vector[vectorIndex] & 0xFE) ^
          ((input[strIndex] >> shiftNum) & 1);
      vectorIndex++;                 //increment vector position
    }
    if(input[strIndex] == char(7))   //end loop if end reached
```

```
      break;
  }
}

void writeNewPixels(FILE * outfile, int* vector, int h, int w)
{
  int elementCount = 17;              //for formatting purposes
  for(int x = 0; x < (h * w); x++)    //runs through all pixels
  {
    for(int y = 100; y > 1; y /= 10) //formatting
    {
      if(vector[x] / y == 0)
        fprintf(outfile, " ");
    }

    fprintf(outfile, "%i ", vector[x]); //write pixel

    elementCount--;

    if(elementCount == 0)             //detect when to end line
    {
```

```
      fprintf(outfile, "\n");
      elementCount = 17;
    }
  }
}
```

File 2, destegFr.cpp, is an extraction program for P2 or P3 PGM files with data hidden by stegFr.cpp.

```
//William Barratt
//Copyright 2002 William Barratt
//destegFr.cpp
/* User-friendly steganography decoding program for .pgm files
   Companion to stegFr.cpp */

#include <iostream.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

void handleArgs(int argc, char* argv[], FILE *& infile);
void readThroughHeaders(FILE * infile, int & h, int & w);
void readIntoVector(FILE * infile, int* vector);
```

```
void interpretVector(int* vector, char * msg);

void outputMessage(char * msg);


int main(int argc, char* argv[])

{

  FILE * infile;                     //source file


  handleArgs(argc, argv, infile);    //get filename


  int h, w;                          //image height, width


  readThroughHeaders(infile, h, w);  //deal with headers


  int * vector = new int[h*w];       //vector for pixel values


  readIntoVector(infile, vector);    //read pixel values


  fclose(infile);                    //close source file


  char * msg = new char[h*w/7];      //string for message
```

```
  interpretVector(vector, msg);       //decode message

  outputMessage(msg);                 //print message to screen

  return 0;
}

void handleArgs(int argc, char * argv[], FILE *& infile)
{
  char* file = new char[80];          //temporary string

  if(argc > 2)                        //if user enters 2 things
  {
    cout << "Too many command-line arguments; aborting."
         << endl;
    exit(3);
  }
  else if(argc == 2)                  //if user enters 1 filename
  {
    infile = fopen(argv[1], "r");
  }
```

```
  else if(argc == 1)                //if user enters no name
  {
    cout << "Name of source image file: ";
    cin >> file;
    infile = fopen(file, "r");
  }
}

void readThroughHeaders(FILE * infile, int & h, int & w)
{
  char* str = new char[80];         //string for reading

  fgets(str, 80, infile);           //reads first line

  if(str[0] != 'P' || str[1] != '2') //check header
  {
    fclose(infile);

    cout << "File header indicates wrong file type" << endl
         << "File must be .pgm with 'P2' in first line" << endl;
```

```
  exit(1);
}

fgets(str, 80, infile);            //second line (comments)
while(str[0] == '#')
{
  fgets(str, 80, infile);          //read through comments
}

  //retrieval of width and height
char* width = new char[80];
char* height = new char[80];

int spaceLoc = strcspn(str, " ");  //finds space between w & h
for(int x = 0; x < spaceLoc; x++)
{
  width[x] = str[x];
}
w = atoi(width);                   //convert to int

height = strstr(str, " ");         //gets height substring
```

```
  h = atoi(height);                  //convert to int

  fgets(str, 80, infile);            //line with max value
}

void readIntoVector(FILE *infile, int* vector)
{
    //identical to like-named function of steg.cpp
    //
  char* line = new char[80];         //buffer for reading lines
  char* tmpstr = new char[80];       //temp for manipulation

  int place = 0;                     //vector index

  while(!feof(infile))               //while not at file end
  {
    fgets(line, 80, infile);         //get one line
    tmpstr = strtok(line, " ");      //gets pixel value

    while(tmpstr != NULL)            //while not at line end
    {
```

```
      if(strcspn(tmpstr, "0123456789") != strlen(tmpstr))
      {                              //checks for numbers in substring
        vector[place] = atoi(tmpstr);//places number in vector
        place++;                     //increments index
      }
      tmpstr = strtok(NULL, " ");    //find next value
    }
  }
}

void interpretVector(int * vector, char * msg)
{
  char tmp = 0;                      //buffer char.

  int vectorIndex = 0, strIndex = 0, shiftNum;

  while(tmp != char(7))              //while end not reached
  {
    tmp = 0;                         //reset buffer

      //runs through, bit-masking etc.
```

```
    for(shiftNum = 6; shiftNum >= 0; shiftNum--)
    {
      tmp = tmp ^ ((vector[vectorIndex] & 1) << shiftNum);
      vectorIndex++;                  //increment place in vector
    }
    if(tmp != char(7))
      msg[strIndex] = tmp;            //set output char (if not 7)
    strIndex++;                       //increment place in string
  }
}


void outputMessage(char * msg)
{
  cout << "The message extracted from the file is:"
       << endl << msg << endl;
}
```