Malware Images: Visualization and Automatic Classification

Group Members

S No	NAME	REG NO
1	Rahul Kinnera	16BCI0021
2	Aditya Sheth	16BCI0034
3	Harshit Mahajan	16BCI0154
4	Sai Teja	16BCI0209

Submitted to

Prof. C. Siva Shanmugam

Course – BCI4003 Malware Analysis

B. Tech

Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Vellore-632014, Tamil Nadu, India

CERTIFICATE

This is to certify that the Project work entitled **“Malware Analyser 3.3 Tool Exploration”** that is being submitted by **Rahul Kinnera, Aditya Seth, Harshit Mahajan and Sai Teja** in B. Tech for the subject **BCI4003 –Malware Analysis** is a record of bonafide work done under my supervision. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted for any other course.

Signature of faculty:

PROF. SIVA SANMUGUM G

Base Paper - **Malware Images: Visualization and Automatic Classification**

ABSTRACT:

We are going to study the techniques to visualize the behaviour of malicious software. Our main aim is to help human analysts to quickly assess and classify the nature of a new malware sample. We are going to explore two visualization techniques: treemaps and thread graph. We hope that both techniques can effectively support a human analyst (a) in detecting maliciousness of software, and (b) in classifying malicious behaviour. [5]

Classifying malware using Image Processing Techniques. **Malware binaries** are visualized as **gray-scale images** with the observation that for many malware families, the images belonging to the same family appear very similar in layout and texture. Neither disassembly nor code execution is required for classification.

Anti-malware software producers are continually challenged to identify and counter new malware as it is released into the wild. A dramatic increase in malware production in recent years has rendered the conventional method of manually determining a signature for each new malware sample untenable. This paper presents a scalable, automated approach for detecting and classifying malware by using pattern recognition algorithms and statistical methods at various stages of the malware analysis life cycle. [12] Our framework combines the static features of function length and printable string information extracted from malware samples into a single test which gives classification results better than those achieved by using either feature individually. In our testing we input feature information from close to 1400 unpacked malware samples to a number of different classification algorithms. Using k-fold cross validation on the malware, which includes Trojans and viruses, along with 151cleanfiles, we achieve an over all classification accuracy of over 98%. [10]

We argue that visual analysis of binary data objects such as data files, process memory, and file systems presented as grayscale graphical depictions helps distinguish structurally different regions of data and thus facilitates a wide range of analytic tasks such as fragment classification, file type identification, location of regions of interest, and other tasks that require an understanding of the primitive data types the objects contain. We believe that, due to the high visual value of this data presentation, such visual analysis is an invaluable help in low-level study of binary data objects and in understanding their structure, and that tools for such visual analysis belong in the toolkit of every researcher studying binary data. In an effort to facilitate development of such tools, this paper presents a visual study of binary fragments created by common kinds of software, and offers a descriptive taxonomy of primitive binary fragments and their graphical depictions. Although significant research has gone into the study of binary fragments, the depth and breadth of this study to date has been limited. Thus, the primary contribution of this paper is an extensible and visual taxonomy to assist and inform researchers conducting low-level analysis of binary objects. [9]

While navigating in an environment, a vision system has to be able to recognize where it is and what the main objects in the scene are. In this paper we present a contextbased vision system for place and object recognition. The goal is to identify familiar locations (e.g., office 610, conference room 941, Main Street), to categorize new environments (office, corridor, street) and to use that information to provide contextual priors for object recognition (e.g., tables are more likely in an office than a street). We present a low-dimensional global image representation that provides relevant information for place recognition and categorization, and show how such contextual information introduces strong priors that simplify object recognition. We have trained the system to recognize over 60 locations (indoors and outdoors) and to suggest the presence and locations of more than 20 different object types. The algorithm has been integrated into a mobile system that provides realtime feedback to the user. [11]

While navigating in an environment, a vision system has to be able to recognize where it is and what the main objects in the scene are. In this paper we present a contextbased vision system for place and object recognition. The goal is to identify familiar locations (e.g., office 610, conference room 941, Main Street), to categorize new environments (office, corridor, street) and to use that information to provide contextual priors for object recognition (e.g., tables are more likely in an office than a street). We present a low-dimensional global image representation that provides relevant information for place recognition and categorization, and show how such contextual information introduces strong priors that simplify object recognition. We have trained the system to recognize over 60 locations (indoors and outdoors) and to suggest the presence and locations of more than 20 different object types. The algorithm has been integrated into a mobile system that provides realtime feedback to the user. [19]

Anti-malware software producers are continually challenged to identify and counter new malware as it is released into the wild. A dramatic increase in malware production in recent years has rendered the conventional method of manually determining a signature for each new malware sample untenable. This paper presents a scalable, automated approach for detecting and classifying malware by using pattern recognition algorithms and statistical methods at various stages of the malware analysis life cycle. Our framework combines the static features of function length and printable string information extracted from malware samples into a single test which gives classification results better than those achieved by using either feature individually. In our testing we input feature information from close to 1400 unpacked malware samples to a number of different classification algorithms. Using k-fold cross validation on the malware, which includes Trojans and viruses, along with 151cleanfiles, we achieve an over all classification accuracy of over 98%. [20]

INTRODUCTION

The field of malicious software (malware), to which worms, bots and other unpleasant artifacts belong, is one of the most active and also one of the most threatening areas of computer security. In recent years, we are observing a huge increase in the number of malware samples collected by anti-virus vendors [1]. Therefore, it is mandatory that we develop tools and techniques to analyse new malware samples with no (or only limited) human interaction.

Static analysis does not allow a high degree of automation during analysis. An alternative to static analysis is dynamic analysis. In dynamic malware analysis, the behaviour of malware is analysed, for example, by executing it within a debugger. One of the most promising approaches to the dynamic analysis of malware consists of sandbox solutions of which several ones have been developed in recent years [2, 3]. A sandbox executes a malware sample in a controlled environment and records all system-level behaviour such as modifications of the file system or the registry. As a result, the sandbox generates an analysis report summarizing the observed behaviour of the sample. In contrast to static analysis, sandbox-based dynamic analysis can be automated to a high degree.

Sandbox reports, often consisting of several hundred entries, usually offer a level of detail that allows careful analysis but overwhelm a human analysts who is interested only in a quick assessment of a particular sample. Given the fact that many samples are just small variants of a well-known malware family, there is a clear need to make the behaviour of malware samples more and easier accessible to a human analyst. In this paper, we will explore a combination of abstraction and visualization to do exactly this. [15]

Binary data is central to the modern computing paradigm and exists in large, complex objects such as process memory, file systems, network flows, as well as in data and executable files. In many cases, however, these objects are heterogeneous, consisting of many dissimilar elements created by commonly used software libraries and processes or algorithms (such as various forms of compression). We call these dissimilar elements primitive binary fragments. Even though the variety of data types is theoretically limitless, in practice they mostly get created by a limited number of popular tools for an equally limited number of consuming programs. Thus, most fragments of complex binary objects weare likely to encounter can be categorized into a number of recursively refined categories, a taxonomy. [9]

Malicious software, malware for short, is a significant threat to the Internet as well as to the computing community. Furthermore, the anti-detection techniques employed by malware are becoming more and more sophisticated. Malware is now being driven by organized crime, with the two most common payloads being the theft of personal information and the recruitment of computers to botnets. Until recently, developers of anti-malware software used manual means of recognizing and classifying malware. However, the enormous increase in volume now forces them to rely heavily on automatic malware analysis, detection and categorization techniques. Thus, in the last ten years or so, researchers have searched for reliable automated malware classification methods. A substantial amount of recent research work has focused on development of tools for collecting, monitoring and run-time analysis of malware [1, 13, 2, 5, 6, 7, 8, 9, 11, 12, 16, 14]. An added complication is that current sophisticated clean software (cleanware) has many of the features of malware (using library calls, containing executables and so on); thus, it is critically important to produce techniques which reliably distinguish cleanware from malware. Thus, we have included a large sample of cleanware in our experimental data.[10]

A vision system that can tell where it is and what it is looking at as it moves through the world. This problem is very difficult and is largely unsolved. Our approach is to exploit visual context, by which we mean a low-dimensional representation of the whole image (the “gist” of the scene). Such a representation can be easily computed without having to identify specific regions or objects. Having identified the overall type of scene, one can then proceed to identify specific objects within the scene. This is the kind of view commonly studied in the object recognition community. The recognition of the object as a coffee machine relies on knowing detailed local properties (its typical shape, the materials it is made of, etc. Object recognition in context is based on our knowledge of scenes and how objects are organized. The recognition of the scene as a kitchen reduces the number of objects that need to be considered, which allows us to use simple features for recognition. Furthermore, the recognition of this scene as a particular kitchen (here, the kitchen of our lab) further increases the confidence about the identity of the object. Hence place recognition can be useful for object recognition.[11]

Seminal conceptions in computational vision have portrayed scene recognition as a progressive reconstruction of the input from local measurements (edges, surfaces), successively integrated into decision layers of increasing complexity. In contrast, some experimental studies have suggested that recognition of real world scenes may be initiated from the encoding of the global configuration, ignoring most of the details and object information.Computational and experimental schools achieve different objectives of recognition: for the former, recognition is a reconstruction procedure of the 3D sceneproperties that is an essential step in tasks involving movement or grasping. For the latter, recognition of the scene implies providing information about the semantic category and the function of the environment.[16]

PROPOSED METHODOLOGY

Step -1: Text : The primary subcategories of the text type include: written language, programming language source code, and markup language. Each of these types is commonly encoded using ASCII or Unicode and may be compressed or encrypted as well.

Step -2 : Audio,Image and Video Media : Multimedia fragments are divided into three major categories: audio, image, and video. Whereas there are common techniques for storing multimedia data without compression, compression using a variety of algorithms is common.

Step -3 : Application: Application fragments are divided into three main categories: machine code, data structure, and packed. Machine Code: Machine language instructions are commonly found within executable files, but may be of particular interest when they exist within data files, such as a Microsoft Word document containing executable code. Machine code may vary depending on the target processor as well as the compiler or assembler that generated the code.

Step – 4:Random : The random primitive fragment type is a sequence of random values and includestwosubcategories, high quality and pseudorandom. Highqualityrandom sequences, such as those derived from atmospheric noise, exhibit no discernible pattern. Pseudorandom numbers may seem random, but are actually deterministic in nature.

Step – 5:Repeating Values : Repeating values are short sequences of byte values that are repeated a large number of times. For example, some file formats align data regions to fixed offsets by adding a sequence of null byte values until the desired offset is reached.

Step – 6 : Encryption,Compression and Encoding : A useful way to think about binary fragments is in terms of generation and transformation. Most of the previous categories included base content, which was encoded in some fashion (US-ASCII, Unicode, PCM, opcodes mapped to byte values, tables of binary coded decimal, and so forth). These binary representations may be further transformed, sometimes repeatedly, in a variety of ways including encryption, compression, and encoding.

We will first present a parameterized method to abstract sandbox reports into increasingly simple summaries. Then we use classical techniques to visualize the resulting reports and make them accessible to the human reader. We claim that the combination of both techniques offers unique insights into the main features of malware such that humans are effectively supported in detection malicious behaviour when it occurs. Furthermore, we claim that the visual representation allows an analyst to rapidly classify the behaviour of new malware samples if it belongs to one of the large and well-known families of malware families currently spreading in the internet.

We use two visualization techniques: tree mapping and thread graphs. Tree mapping [4, 5] displays the distribution of the individual operations performed by a sample. The resulting tree map presents this information as a set of nested rectangles and provides a quick overview of the main overall behaviour of the sample, e.g., whether the main task of the sample lies in the area of network interaction, changes to the file system, or interaction with other processes. Since tree maps display nothing about the sequence of operations, we use thread graphs to visualize the temporal behaviour of the individual threads of a sample. A thread graph can be regarded as a behavioral fingerprint of the sample. An analyst can then study this behaviour graph to quickly learn more about the actions of each individual thread.[4]

The methodology used in our experiment is given below. We used the k-fold testing technique with k =5 .

1. Extract the features based on both FLF and PSI techniques.

2. Combine the features vectors and make a WEKA(arff) file.

3. Build families Fs for s = i =1...13

4. Select a family Fi and split it into 5 parts.

5. Select the same instances of Fj from other families Fj(j = i) and split into 5 parts in the same manner as for Fi.

6. Build 5 sets ofTraining Set Data and Test Set Data in respective portions of 80% and 20%.

7. Select features from the Training Data.

8. Select the classifier C to be used.

9. Call WEKA libraries to train the C using the Training Data.

10. Evaluate the Test Set Data.

11. Repeat four more times.

12. Repeat for other classifiers.

13. Repeat for other classifiers.[10]

Identify the flat view of man made environment vertically structured.Small space with large elements.Categorize Man made open environment and man made closed urban environment.Then divide Flat view of a man made closed urban environment such that large space with small elements.Then integrate perspective view of a man made open environment.[17]

IMPLEMENTATION PROCEDURE

We will use a database of malware samples collected using honeypots. To analyse these samples, we first execute them in a controlled, instrumented environment (sandbox) and observe their behaviour during runtime. In the second step, we visualize the collected information using different techniques. As an output, our technique generates different visualizations of the sample's behaviour, which enable a human analyst to get a quick overview of what a given malware sample does.[4]

Step-1:Byte Plot the image or text

Step-2 :Insert two 5 MB's on the image

Step-3 :Convert into ASCII Text

Step-4:Using Data Structure on the image or text

Step-5:Form Compressed Image 1

Step-6:Divide into different Compressed Image N

Step-7:Form Unicode URLs

Step-8:Then apply a complex Data Structure [8]

Step -1 : Identify the global image features in the taken object

Step -2: Using mathematical equation find the place recognition

(i) Werable test bed

(ii)Model for place recognition

(iii)Performance of place recognition

(iv) Categorizing novel places

Step -3: From scenes to objects

(i) Contextual priming for object detection

(ii) Contextual priors for object localization[12]

Step-1 : Level the scene into Description and Scene Models

Step-2 : Divide into Subordinate Level,Basic level and Superordinate level.

Step 3: Identify the (i)degree of openness

(ii)degree of expansion

(iii)degree of roughness[19]

CODE:

```
FiletoBinary.java
import java.io.*;
public class FiletoBinary {
       static StringBuilder sb;
       public static void main(String[] args)
       {

              try {
                  StringBuilder sb = new StringBuilder();
                  File file = new File("D:\\Project1\\test\\mal.txt");
                  DataInputStream input = new DataInputStream( new FileInputStream( file
) );
                  try {
                      while( true )
```

```
                              {
                              sb.append( Integer.toBinaryString( input.readByte() ) );
                           }
                       } catch( EOFException eof ) {
                       } catch( IOException e ) {
                           e.printStackTrace();

                       }
                       System.out.println(sb.toString());
                   } catch( FileNotFoundException e2 ) {
                       e2.printStackTrace();
                   }
           }
    }
```

Image.java

```
import java.io.File;
import java.io.*;
import java.io.IOException;
import java.util.Scanner;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
public class Image{
public static void main(String args[])throws IOException{

     int width = 640;
     int height = 320;

     BufferedImage img = new BufferedImage(width, height, BufferedImage.TYPE_INT_ARGB);

     File f = null;
    //ImagetoBinary il = new ImagetoBinary();
     String bin1 =
"11000011110011101001011000011101110111001111011111101101110010111001101100101";
     char[] bin = bin1.toCharArray();

     FileInputStream fstream = new FileInputStream("D:\\Project1\\test\\a.txt");
     DataInputStream in = new DataInputStream(fstream);
     BufferedReader br = new BufferedReader(new InputStreamReader(in));
     String data;
     String[] tmp=null;
     while ((data = br.readLine()) != null)   {
       tmp = data.split(" ");
     }
     int[] a=new int[tmp.length];
     for(int i=0;i<tmp.length;i++)
     {
       String n = tmp[i];
       int z = Integer.parseInt(n);
       a[i]=z;
     }

     fstream = new FileInputStream("D:\\Project1\\test\\b.txt");
     in = new DataInputStream(fstream);
     br = new BufferedReader(new InputStreamReader(in));
     tmp=null;
     while ((data = br.readLine()) != null)   {
       tmp = data.split(" ");
```

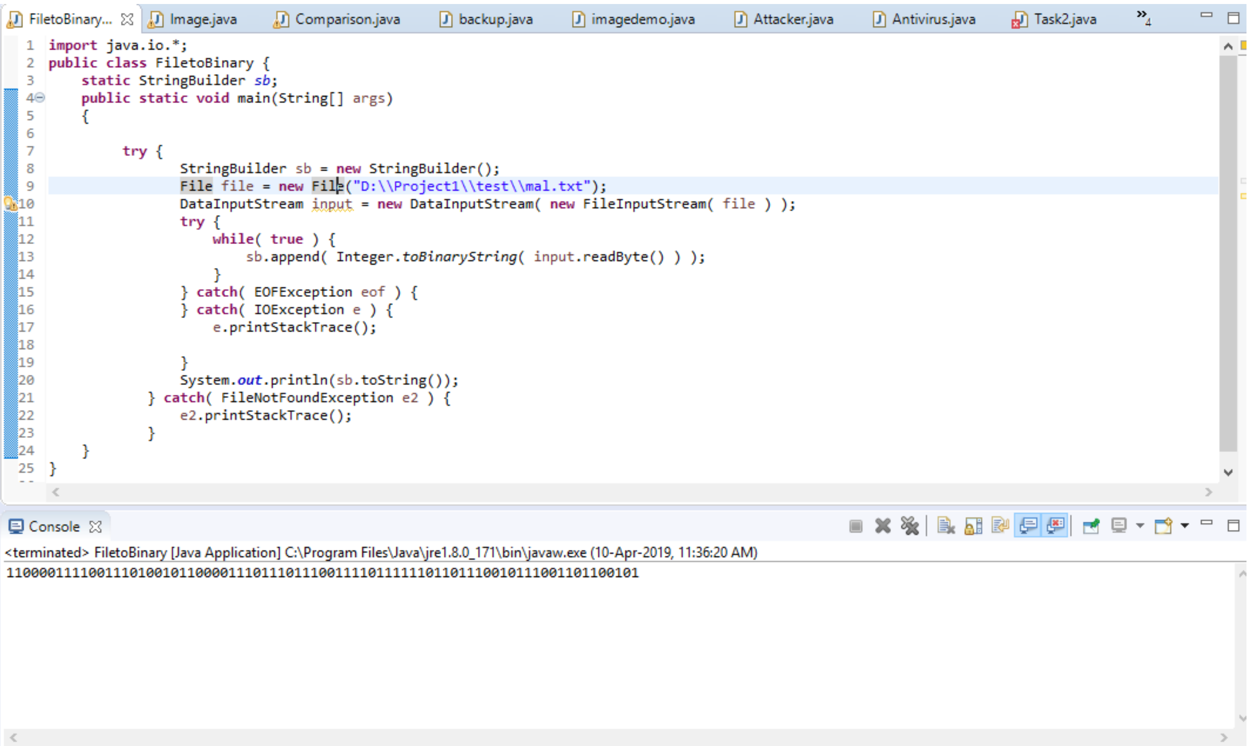
```
  }
int[] b=new int[tmp.length];
for(int i=0;i<tmp.length;i++)
{
   String n = tmp[i];
   int z = Integer.parseInt(n);
   b[i]=z;
}

fstream = new FileInputStream("D:\\Project1\\test\\g.txt");
in = new DataInputStream(fstream);
br = new BufferedReader(new InputStreamReader(in));
tmp=null;
while ((data = br.readLine()) != null)   {
   tmp = data.split(" ");
}
int[] g=new int[tmp.length];
for(int i=0;i<tmp.length;i++)
{
   String n = tmp[i];
   int z = Integer.parseInt(n);
   g[i]=z;
}


int count=0;
int i=0;
int m=0,n=0,l=0;
for(int y = 0; y < height; y++){
  for(int x = 0; x < width; x++){
    int c = (int)a[m];
    m++;
    int r;
    if(count>=bin.length)
    {
        //r = (int)((Math.random()<0.5)?0:1);
         r=0;
    }
    else
         {
                r = bin[i];
                count++;
                i=i+1;
         }
    int d = (int)b[n];
    n++;
    int e = (int)g[l];
    l++;

    int p = (c<<24) | (r<<16) | (d<<8) | e;

    img.setRGB(x, y, p);
  }
}

try{
  f = new File("D:\\Project1\\Output\\Output.png");
  ImageIO.write(img, "png", f);
  System.out.println("Sucessfully Created");
```

```
      }catch(IOException e){
       System.out.println("Error: " + e);
     }
  }
}
```

Comparison.java

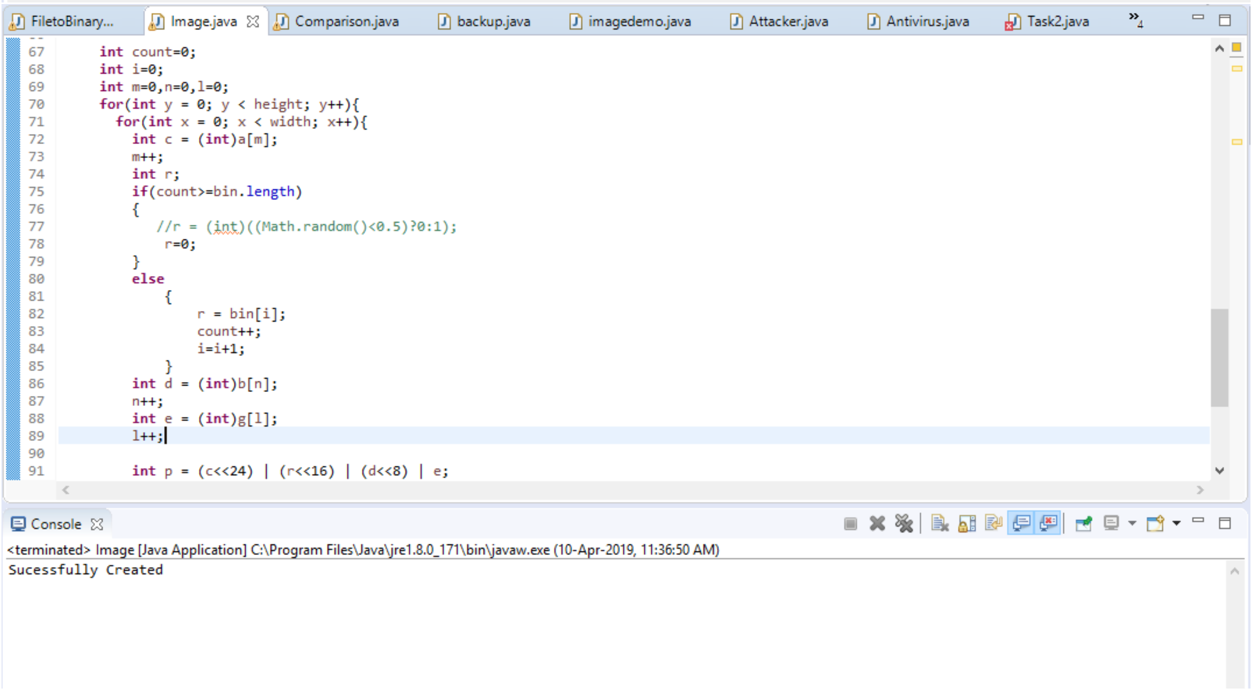
```
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.io.File;
import java.io.IOException;

class Comparison
{
    public static void main(String[] args)
    {
        BufferedImage imgA = null;
        BufferedImage imgB = null;
        File fileA=null, fileB=null;
        boolean flag=true;
        File folder = new File("D:\\Project1\\Malware");
        File[] listOfFiles = folder.listFiles();
        double[] diff = new double[listOfFiles.length];
        for (int i = 0; i < listOfFiles.length; i++)
        {
             fileB = new File("D:\\Project1\\Malware\\"+listOfFiles[i].getName());
        try
        {
            fileA = new File("D:\\Project1\\Output\\Output.png");
            imgA = ImageIO.read(fileA);
            imgB = ImageIO.read(fileB);
        }
        catch (IOException e)
        {
            System.out.println(e);
        }
        int width1 = imgA.getWidth();
        int width2 = imgB.getWidth();
        int height1 = imgA.getHeight();
        int height2 = imgB.getHeight();

        if ((width1 != width2) || (height1 != height2))
            System.out.println("Error: Images dimensions"+
                                             " mismatch");
        else
        {
            long difference = 0;
            for (int y = 0; y < height1; y++)
            {
                for (int x = 0; x < width1; x++)
                {
                    int rgbA = imgA.getRGB(x, y);
                    int rgbB = imgB.getRGB(x, y);
                    int redA = (rgbA >> 16) & 0xff;
                    int greenA = (rgbA >> 8) & 0xff;
                    int blueA = (rgbA) & 0xff;
                    int redB = (rgbB >> 16) & 0xff;
                    int greenB = (rgbB >> 8) & 0xff;
                    int blueB = (rgbB) & 0xff;
                    difference += Math.abs(redA - redB);
                    difference += Math.abs(greenA - greenB);
                    difference += Math.abs(blueA - blueB);
                }
            }

            // Total number of red pixels = width * height
            // Total number of blue pixels = width * height
            // Total number of green pixels = width * height
            // So total number of pixels = width * height * 3
            double total_pixels = width1 * height1 * 3;

            // Normalizing the value of different pixels
            // for accuracy(average pixels per color
            // component)
            double avg_different_pixels = difference / total_pixels;

            // There are 255 values of pixels in total
            double percentage = (avg_different_pixels / 255) * 100;

            diff[i]=percentage;
            //System.out.println("Difference Percentage-->" + percentage);
            /*if(percentage>0.0653)
            {
             System.out.println("The file is "+(fileB.getName().replaceFirst("[.][^.]+$", 
"")));
             flag = false;
            }*/

        } }
        double min = diff[0];
        int index=0;
        for(int i=0;i<listOfFiles.length;i++)
        {
             if(min>diff[i])
             {
                    min=diff[i];
                    index=i;
             }
        }
        for(int i=0;i<listOfFiles.length;i++)
        {
             if(index==i)
             {
                    fileB = new File("D:\\Project1\\Malware\\"+listOfFiles[i].getName());
                    System.out.println("The file may be 
"+(fileB.getName().replaceFirst("[.][^.]+$", "")));
             }
        }
        /*if(flag)
        {
             System.out.println("Not a malware");
        }*/
    }
}
```

OUTPUT:



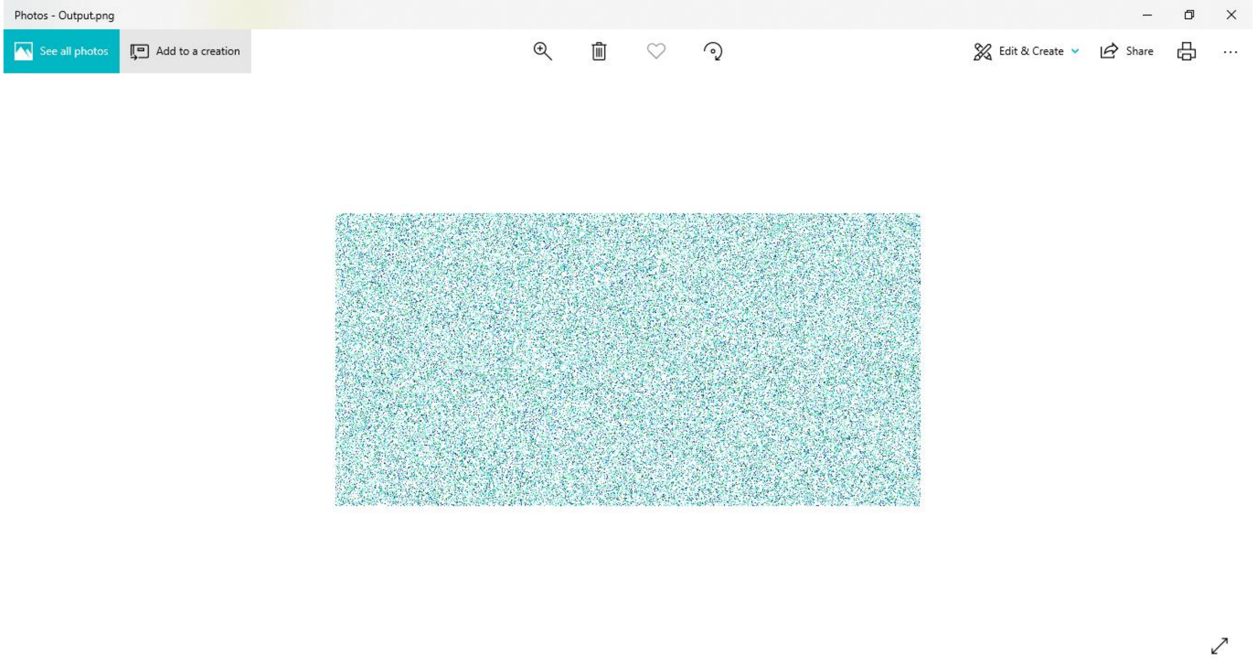
```java
import java.io.*;
public class FiletoBinary {
    static StringBuilder sb;
    public static void main(String[] args)
    {

         try {
              StringBuilder sb = new StringBuilder();
              File file = new File("D:\\Project1\\test\\mal.txt");
              DataInputStream input = new DataInputStream( new FileInputStream( file ) );
              try {
                  while( true ) {
                      sb.append( Integer.toBinaryString( input.readByte() ) );
                  }
              } catch( EOFException eof ) {
              } catch( IOException e ) {
                  e.printStackTrace();

              }
              System.out.println(sb.toString());
          } catch( FileNotFoundException e2 ) {
              e2.printStackTrace();
          }
    }
}
```

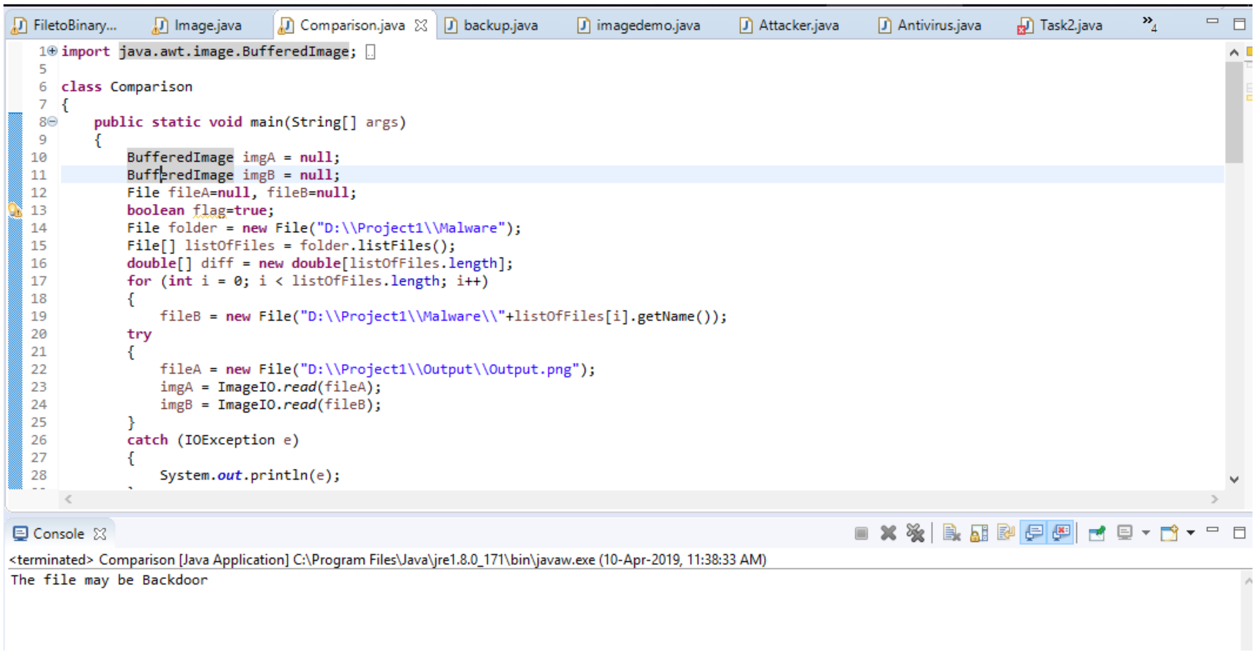
```
<terminated> FiletoBinary [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (10-Apr-2019, 11:36:20 AM)
11000011110011101001011000011101110111001111011111101101110010111001101100101
```



```java
        int count=0;
        int i=0;
        int m=0,n=0,l=0;
        for(int y = 0; y < height; y++){
          for(int x = 0; x < width; x++){
            int c = (int)a[m];
            m++;
            int r;
            if(count>=bin.length)
            {
               //r = (int)((Math.random()<0.5)?0:1);
                r=0;
            }
            else
                {
                    r = bin[i];
                    count++;
                    i=i+1;
                }
            int d = (int)b[n];
            n++;
            int e = (int)g[l];
            l++;

            int p = (c<<24) | (r<<16) | (d<<8) | e;
```

```
<terminated> Image [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (10-Apr-2019, 11:36:50 AM)
Sucessfully Created
```





```java
import java.awt.image.BufferedImage;

class Comparison
{
    public static void main(String[] args)
    {
        BufferedImage imgA = null;
        BufferedImage imgB = null;
        File fileA=null, fileB=null;
        boolean flag=true;
        File folder = new File("D:\\Project1\\Malware");
        File[] listOfFiles = folder.listFiles();
        double[] diff = new double[listOfFiles.length];
        for (int i = 0; i < listOfFiles.length; i++)
        {
            fileB = new File("D:\\Project1\\Malware\\"+listOfFiles[i].getName());
        try
        {
            fileA = new File("D:\\Project1\\Output\\Output.png");
            imgA = ImageIO.read(fileA);
            imgB = ImageIO.read(fileB);
        }
        catch (IOException e)
        {
            System.out.println(e);
```

```
<terminated> Comparison [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.exe (10-Apr-2019, 11:38:33 AM)
The file may be Backdoor
```

CONCLUSION:

A novel approach to malware analysis based on visualizing and processing malware as images. A commonly used image feature descriptor is used to characterize the malware globally. The preliminary results are very encouraging, with high accuracy classification that is competitive with the state of the art results in the literature at a significantly less computational cost. We believe that using computer vision techniques for malware analysis opens the path to a broader spectrum of novel ways to analyze malware

REFERENCES

[1] Symantec. Internet Security Threat Report Volume XIV. Internet: http://www.symantec.com/business/theme.jsp?themeid= threatreport, April 2009.

[2] U. Bayer. Anubis: Analyzing Unknown Binaries. http:// analysis.seclab.tuwien.ac.at/.

[3] Norman ASA. Norman SandBox. http://sandbox.norman.no/.

[4] Ben Shneiderman. Treemaps for space-constrained visualization of hierarchies. Internet: http://www.cs.umd.edu/hcil/ treemap-history/.

[5] Ben Shneiderman. Tree Visualization With Tree-Maps: 2-D SpaceFilling Approach. ACM Trans. Graph., 11(1), 1992.

[6] Conti, G. Bratus, S. Shubina, A. Lichtenberg, A. Ragsdale, R. Perez-Alemany, R. Sangster, B. and Supan, M. 2010. A Visual Study of Binary Fragment Types *Black Hat USA*

[7]. Multi-dimensional Scaling, Dr Toolbox http://homepage.tudelft.nl/19j49/Matlab_Toolbox_for_Dimensionality_Reduction.html

[8] Torralba, A. Murphy, K.P. Freeman, W.T. and Rubin, M.A. 2003, Context-based vision systems for place and object recognition, *Intl. Conf. on Computer Vision (ICCV).*

[9]. Oliva, A. and Torralba. A, 2001. Modeling the shape of a scene: a holistic representation of the spatial envelope, *International Journal of Computer Vision,* Vol. 42(3), 145-175.

[10]. Symantec Global Internet Security Threat Report, April 2010.

[11]. Anubis: Analyzing Unknown Binaries, < http://anubis.iseclab.org/>

[12]. Yoo, I. Visualizing Windows Executable Viruses Using Self-Organizing Maps., 2004 *International Workshop on Visualization for Cyber Security (VizSec)..*

[13]. Quist, D.A. and Liebrock, L.M. 2009. Visualizing compiled executables for malware analysis. *International Workshop on Visualization for Cyber Security (VizSec)*, 27-32.

[13]. Karim, M. E., Walenstein, A., Lakhotia, A. & Parida, L. 2005. Malware phylogeny generation using permutations of code. *Journal in Computer Virology,* **1** (1)**:**13-23.

[14]. Kolter, J. Z. and Maloof, M. A. 2004. Learning to detect malicious executables in the wild. *International Conference on Knowledge Discovery and Data Mining*, 470-478.

[15]. Gao, D., Reiter, M. K. & Song, D. 2008. Binhunt: Automatically finding semantic differences in binary programs. *Information and Communications Security*, **5308**:238–255,

[16] Tian, R. Batten, L.M. and Versteeg. S.C. 2008. Function length as a tool for malware classification. *3rd International Conference on Malicious and Unwanted Software (MALWARE)*.

[17] W. Li, K. Wang, S. Stolfo, and B. Herzog. Fileprints: Identifying File Types by n-gram Analysis. IEEE Information Assurance Workshop, 2005.

[18] M. Karresand and N. Shahmehri. File Type Identification of Data Fragments by Their Binary Structure. IEEE Information Assurance Workshop, 2006.

[19] M. McDaniel and M. Heydari. Content Based File Type Detection Algorithms. Hawaii International Conference on System Sciences, 2003.

[20] G. Hall and W. Davis. Sliding Window Measurement for File Type Identification. ManTech Cyber Solutions International White Paper.