Reverse Engineering and Analysis

*Abstract*—Reverse engineering is the most popular malware analysis method specified in android development. This lab provides a malware application and teaches students how to analyse the application and how to protect the mobile device by using reverse engineering from the malware application.

# Introduction

Reverse engineering is a method to take apart an object to see how it works in order to duplicate or enhance the object, it is a practise taken from older industries, and is now frequently used on computer hardware and software.

Software reverse engineering involves reversing a program’s machine code (string 0’s and 1’s that are sent to the logic processor) back into the source code that it was written in, using program language statements.

There are many tools that implement reverse engineering in android development. It can fully decompile the apps thus making the code readable and understandable.

XdaDeveloperBrut.all has been working on a decompiler for android apps for a while. He had found the first workaround for enabling google maps navigation outside US by decompiling Google maps for android. The tool is called APKTool and is opensource.

Another called dex2jar, developed by a Chinese student will translate dex to jar file. The next step would be to use jd-gui, the source code is readable as dex2jar makes some optimizations.

# LAB ACTIVITY

You will need three tools for this lab: (For your convenience, I have downloaded all these tools needed in this lab and put them on the website.)

Dex2jar: <https://code.google.com/p/dex2jar/>

JD\_GUI: <http://www.onlinedown.net/soft/70298.htm>

Apktool:<https://code.google.com/p/android-apktool/> (apktool\_2.0.1.jar and  [apktool-install-windows-r05-ibot.tar.bz2](https://android-apktool.googlecode.com/files/apktool-install-windows-r05-ibot.tar.bz2))

**Part1: Decompile APK to JavaCode**

This part of the lab requires to two tools to be downloaded dex2jar and JD\_GUI. Dex2jar is a tool to transit classes.dex by converting .apk extention into .jar file . JD-GUI is a decompiler tool.

Suppose you want to analyze MyFirstApp.apk. Open MyFirstApp project in Eclipse, right-click project root, Android Tools – Export Unsigned Application Package to generate MyFirstApp.apk.

Change the extension of the apk file from .apk to .zip, and unpack it with winrar to get classes.dex. Then unpack the tool dex2jar, and place classes.dex into dex2jar folder. Run “dex2jar classes.dex” to get classes\_dex2jar.jar. Next, run JD\_GUI , and open classes\_dex2jar.jar to view the Java source code.

**Part2: Decompile apkfile to get AndroidManifest.xml**

Unpack apktool 2.0.1.tar and  [apktool-install-windows-r05-ibot.tar.bz2](https://android-apktool.googlecode.com/files/apktool-install-windows-r05-ibot.tar.bz2) to get the three following files: aapt.exe, apktool.bat, apktool.jar .

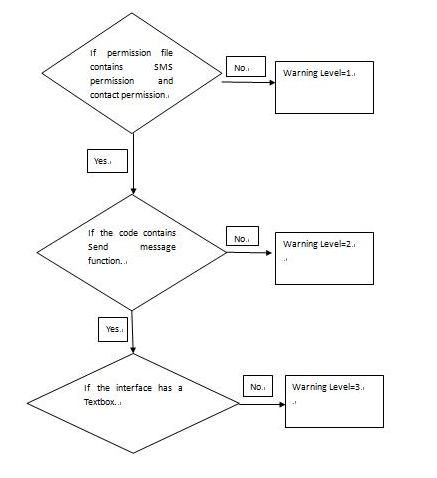
Copy the apktool.jar into the folder containing aapt.exe and apktool.bat. Make a new folder called “tmp” in this folder. Use the command prompt to run the following command to get the .xml file

apktool.bat d -f MyFirstApp.apk -o ./tmp

The AndroidManifest.xml file has been generated in the tmp folder.

**Part3: Analyze the AndroidManifest.xml file with a set of rules**

Here is the basic flowchart for the set of rules:



In Eclipse, File—New—Java Project to create a java project called Auto\_Reverse. Copy and paste the following code to Auto\_Reverse,java:

|  |
| --- |
| //package auto\_reverse;  import java.io.BufferedReader;  import java.io.FileReader;  //import java.io.FileWriter;  import java.io.IOException;  import java.util.Scanner;  import java.util.logging.Level;  import java.util.logging.Logger;  /\*\*  \*  \*/  public class Auto\_Reverse {  /\*\*  \* @param args the command line arguments  \*/  public static void main(String[] args) {  try {  int warninglevel=0;//0 is lowest, 2 is highest.  FileReader fin=new FileReader(args[0]);  BufferedReader br = new BufferedReader(fin);  String Line=br.readLine();  //the rule  int flag = 0;  for (int i=0;Line !=null;i++){  if (Line.contains("android.permission.SEND\_SMS")) flag = flag | 1;  if (Line.contains("android.permission.READ\_CONTACTS")) flag = flag | 2;  Line = br.readLine();  }  br.close();  fin.close();  if (flag == 3){  warninglevel = 1;  FileReader fi=new FileReader(args[1]);  BufferedReader b = new BufferedReader(fi);  String l=b.readLine();  for (int j=0;l !=null;j++)  {  if(l.contains("sendTextMessage")){  warninglevel = 2;  break;  }  l = b.readLine();  }  b.close();  fi.close();  }    System.out.println("The warning level for this AndroidManifest.xml file is:");  System.out.println(warninglevel);  } catch (IOException ex) {  Logger.getLogger(Auto\_Reverse.class.getName()).log(Level.SEVERE, null, ex);  }  }  } |

# Demo

Please include step-by-step screenshots in the lab report.

# Correction

I just realized that the code listing above is buggy. Please change Auto\_Reverse.java to follow the flowchart on the right, as a simple Java programming exercise. To make it user-friendly, pass in AndroidManifest.xml and MainAcvivity.java file as 1st and 2nd arguments to the program, e.g.

Anto\_Reverse ./AndroidManifest.xml ./MainActivity.java

To test the program, you can manually add the permissions and function calls to the xml and java files.

If you cannot finish it before Monday night, it’s OK. Just make a note in the lab report, and finish it in the next lab report.

