A Framework to Investigate Android Permissions for Malware Detection

Shivani Gupta[1], Jaya Dhawan[2]

[1]Haryana Engineering College, Jagadhri

Email: shivani.gupta389@gmail.com

[2] Haryana Engineering College, Jagadhri

***Abstract*— the android security model is based on an authorization and sandbox manner. Each application runs in its own Dalvik Virtual Machine with a unique ID assigned to application. This prevents an application from using information/data of another application. Although Android is most widely used, there exists a lack of applications in order to completely benefit from this operating system. Thus, third party application developers create new applications and launch them in the Android Market. This permits users access to thousands of applications; it is however important that the user needs to totally trust the applications before installing them. It is for this reason that every application publishes the permissions that it requires during installation. The user can either grant all permissions or deny all, in which case, the installation of the application is aborted. In order to distribute these applications Google came up with Android Market. Here users can access both paid and free applications. Every Android phone has this application and hence users can browse and download any application they entail from Android Market. However, there have been many malicious applications published in Android Market. Hence it means a necessity for Google to test each and every application and fresh the Android Market by eradicating malwares. It is also important to see to it that the loopholes and bugs of current applications are not exploited by hackers. One way in which an attacker can entice users to download the malevolent software is by repackaging applications using reverse engineering tools. The attacker changes the code in order to incorporate the spiteful code and repackages the application and publishes them in the app market. Users typically cannot differentiate between the malware application and the legitimate application and thereby end up installing the malware. Reverse Engineering is a process with the aid of which we can discover and understand the complete working of an application by learning its function, structure and functions. In this project the tools we use for reverse engineering are AdvanceApkTool, Dex-Manger, DextoJar and Jd-Gui**

Keywords- Android app, Application PacKage File, APK files structure, Android's four-layer model

1. INTRODUCTION

The most common operating system (OS) for mobile device, as of early 2013, is the Android OS by Google. The Android platform is designed with openness in mind, meaning all of the system's source code is presented for download, modification and review The Google Play Store uses a blacklist style of accepting Android applications ("apps"); so as to is all apps are accepted unless they are reported by users. Android relies on its permissions system in order to lessen the risk of a malicious app on a device. A user can manually check the list of permissions necessary by the app upon installation as a method to determine if it is a legitimate app [1].

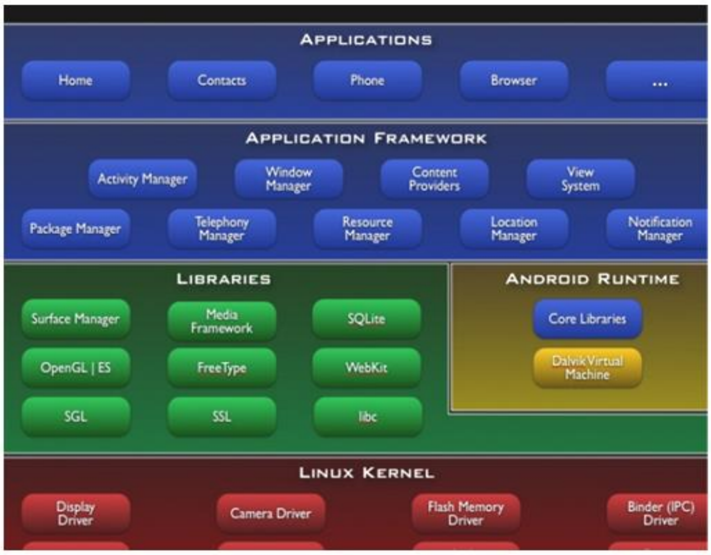


Figure 1 Android's four-layer model [2]

Figure 1 shows the four layers of the Android operating system. The monolithic Linux kernel placed on the lowest layer. It is responsible for process and memory management, handles device drivers and additionally supplies the hardware abstraction layer to other parts of the system. The kernel has been greatly optimized to meet the requirements of a mobile device [2].

1.2 Android app

Nowadays, occupying the largest market share among all varieties of apps, Android apps have drawn increasing attention of people. According to the statistics data from the number of apps from the largest Android distribution market, i.e. GooglePlay has reached to millions level, and is continually sharply Increasing. However, huge app market also attracts attackers who upload elaborate malware without any warning. With the improvement of Android apps, people are used to logging in personal websites, storing private data, and yet paying online through Android smart phones. Private information therefore becomes the goal of attackers. [3] Android is based on Linux kernel where applications run data independently and inter process communication is strictly based on a permission system. Application download requires users to blindly grant access to the listed permissions or deny installation. [4]. Android apps are developed in Java language. Compiled class files are converted in a single executable dex file (dalvik executable) using Dx tool to run under constrained processing and low memory environment. Executable source of an app is stored in dalvik. App is a zip file as shown in figure 1 runs on a register based dalvik virtual machine developed for low memory, and constrained processing embedded environment [6].

1.2.1 Application PacKage File

Android Manifest is binary consisting information such as Package name, Permissions an app would use once installed, Activities running within an app as well as services, receivers and content providers. Resource in application package stores icons, shortcuts, images, dimension constants, string constants, and drawable components. CLASSES is a dalvik executable that stores executable code of the app .Android apps are self-signed by the developers with on third party signature authentication required for development and distribution. All of the above components shown in figure 1 is combined into a single Android PacKage (APK) [6].

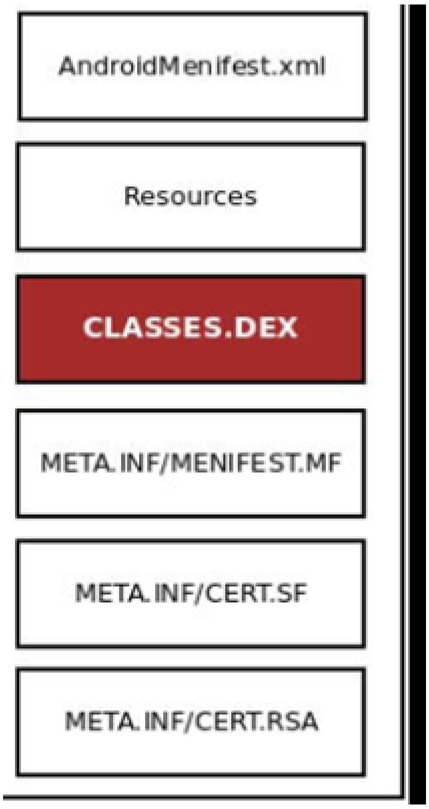


Figure 2.Android PacKage File format [6]

1.3 APK files structure

An APK consists at a minimum, the directories and files shown in Figure 2. This AndroidManifest.xml file is most important in the research. This is stored in a binary XML layout and must be converted to a plain text format before becoming human-readable. This file includes information such as the minimum Android version the app was designed for, the major movement (which is launched upon opening the app) and other details important to the basic functionality of an Android app. Most significantly for our purposes, it contains declarations of the Android permissions the app requires [1].

2. RELATED WORK

The research work performed in this field by diverse researchers is presented as follows:

Quang Doet al. in (2014) [1] Android mobile devices are becoming an admired alternative to computers. The rise in the number of tasks performed on mobile devices means perceptive information is stored on the devices. Consequently, Android devices are a potential vector for scandalous exploitation. Presented research on enhancing user privacy on Android devices can generally be classified as Android alterations. These solutions often involve operating system modifications, which significantly reduce their capability. This research proposes the utilize of permissions removal, wherein a reverse engineering process is used to eliminate an app's permission to a resource. The repackaged app will lope on all devices the original app supported. Findings that are based on a study of seven accepted social networking apps for Android mobile devices indicate that the difficulty of permissions removal may vary among types of permissions and how well-integrated a permission is within an app

Andr´eEgnerset al. in (2012) [2] Permission models have become very frequent on smart phone operating systems to manage the rights granted to installed third party applications (apps). Past to installing an app, the user is typically accessible with a dialog box screening the permissions requested by the app. The user has to choose either to accept all of the requested permissions, or prefer not to proceed with the installation. Most normal users are not able to fully grasp which set of permissions approved to the application is potentially harmful. In addition to the knowledge gap between user and application programmer, the omitted granularity and alterability of most permission model implementations help an attacker to circumvent the permission model. In this paper it focuses on the permission model of Google's Android platform, detail the permission model, and present a collection of attacks that can be composed to fully compromise a user's device using inconspicuously looking applications requesting non-suspicious permits.

ChenkaiGuoet al. in (2015) [3] Attackers who designed malware appear to be so cautious that most of the malware are disguised as normal apps. This brings about huge difficulties to detect the malware. Similar with conventional PC testing, there are two main detection methods for Android malware: static analysis and energetic monitoring. However, these methods inevitably face the challenge of code confusion performance cost. In this paper, a new assessment algorithm based on the statistic technologies is proposed. By extracting permission features, it proposes a sensible method to judge whether an Android app is malicious or not. Besides, an evaluation prototype system MalDetector is developed to confirm the effectiveness of this approach. It took 1260 malware and 10k promotes apps as "malevolent" and "benign" datasets respectively. Adequate experiments on these datasets show that MalDetector is more accurate and with lower false positive rate compared with other traditional methods.

GarimaBajwaet al. in (2015) [4] the intention of an Android application, determined by the source code analysis is used to identify potential maliciousness in that application (app). Similarly, it is possible to analyze the unintentional behaviors of an app to identify and reduce the window of vulnerabilities. Unintentional behaviors of an app can be any developmental loopholes like as software bugs disregarded by a developer or introduced by an adversary intentionally. FindBugsTMand Android Lint are a couple of tools that can detect such bugs easily. A software bug can cause many security vulnerabilities (known or unknown) and vice-versa, thus, creating a many-to-many mapping. In this approach, construct a matrix of mapping between the bugs and the potential vulnerabilities. A software bug detection tool is used to identify a list of bugs and create an empirical list of the vulnerabilities in an app. The many-to-many mapping matrix is obtained by two approaches - sternness mapping and probability mapping. These mappings can be used as tools to measure the unknown vulnerabilities and their strength.

Mario Franket al. in (2012) [5] Android and Facebook provide third-party applications with access to users' private data and the capability to perform potentially sensitive operations (e.g., post to a user's wall or place phone calls). As a safety measure, these platforms restrict applications' privileges with permission systems: users must endorse the permissions requested by applications before the applications can make privacy- or security-relevant API calls .However, current studies have shown that users often do not understand permission requests and lack a notion of typicality of requests. As a _rst step towards simplifying permission systems, it clusters a corpus of 188,389 Android applications and 27,029 Facebook applications to and patterns in permission requests. Using a method for Boolean matrix factorization for ending overlapping clusters, that Facebook permission requests follow a clear structure that exhibits high stability when attend with only clusters, whereas Android applications demonstrate more complex permission requests, also low-reputation applications a lot deviate from the permission request patterns that identified for high-reputation applications signifying that permission request patterns are indicative for user satisfaction or application quality.

ParvezFaruki et al. in (2013) [6] Popularity of Android smart phone has led to exponential increase of sophisticated malware coercion prompting the academia research, security researchers and Anti-Virus (AV) industry to look for smart finding methods to protect user against malware app threat. Statistical signature methods play a vital role to end the malware authors spreading malicious content during apps. Statistical signature is robust against repackaged and code obfuscated malware, accepted app obfuscation techniques. DroidOLytics is a syntactic approach that finds regions of statistical likeness with known malware to detect variants of known malware families.

Wook Shin et al. in (2010) [7] this paper suggests a formal model of the Android permission scheme. It describes the scheme specifying entities and relationships, and supplies a state-based style which includes the behavior specification of permission authorization and the interactions between application components, also shown how we can logically confirm the security of the specified system. Utilizing a theorem proved, it can verify security with given security requirements based on mechanically checked proofs. The projected model can be used as a reference model when the scheme is implemented in a different embedded platform, or when the current scheme with additional constraints extend or elements, demonstrate the use of the verifiable specification through finding a sanctuary vulnerability in the Android system. To this knowledge, this is the first formalization of the permission scheme enforced by the Android structure.

Hamid Bagheriet al. in (2015) [8] Android is the most popular platform for mobile devices. It facilitates division of data and services among applications using a rich inter-app communication system. While access to resources can be restricted by the Android permission system, enforcing permissions is not sufficient to prevent safety violations, as permissions may be mismanaged, intentionally or unintentionally. Android's enforcement of the permissions is at the stage of individual apps, allowing multiple malicious apps to collude and combine their permissions or to trick susceptible apps to perform events on their behalf that are beyond their individual privileges. Present COVERT, a device for compositional analysis of Android inter-app vulnerabilities. COVERT's analysis is modular to permit incremental analysis of applications as they are installed, efficient, and removed. It statically analyzes the reverse engineered source code of every individual app, and extracts relevant safety specifications in a format suitable for formal verification. Given a gathering of specifications extracted in this way, a formal analysis engine(e.g., model checker) is then used to confirm whether it is safe for a grouping of applications—holding certain permissions and potentially interacting with apiece other—to be installed together. This experience with using COVERT to examine over 500real-world apps corroborates its capacity to find inter-app vulnerabilities in bundles of some of the most popular apps on the market.

3. PROPOSED WORK

3.1 Problem Statement

Reverse Engineering is a process with the aid of which we can discover and understand the absolute working of an application by learning its operation, structure and functions. In this project the tools we use for reverse engineering are, ApkTool, DexManager, Dex2Jar, JD-GUI and Android SDK. The application, in its pre-compiled binary layout, is distributed and hence it is not possible to directly debug the source code. However, there are disassemblers that transfer or reverse the Dalvik Bytecode into readable format. The binaries for Dalvik Virtual Machines are in the .dex format. Backsmali is a disassembler that is used for .dex files in Dalvik.

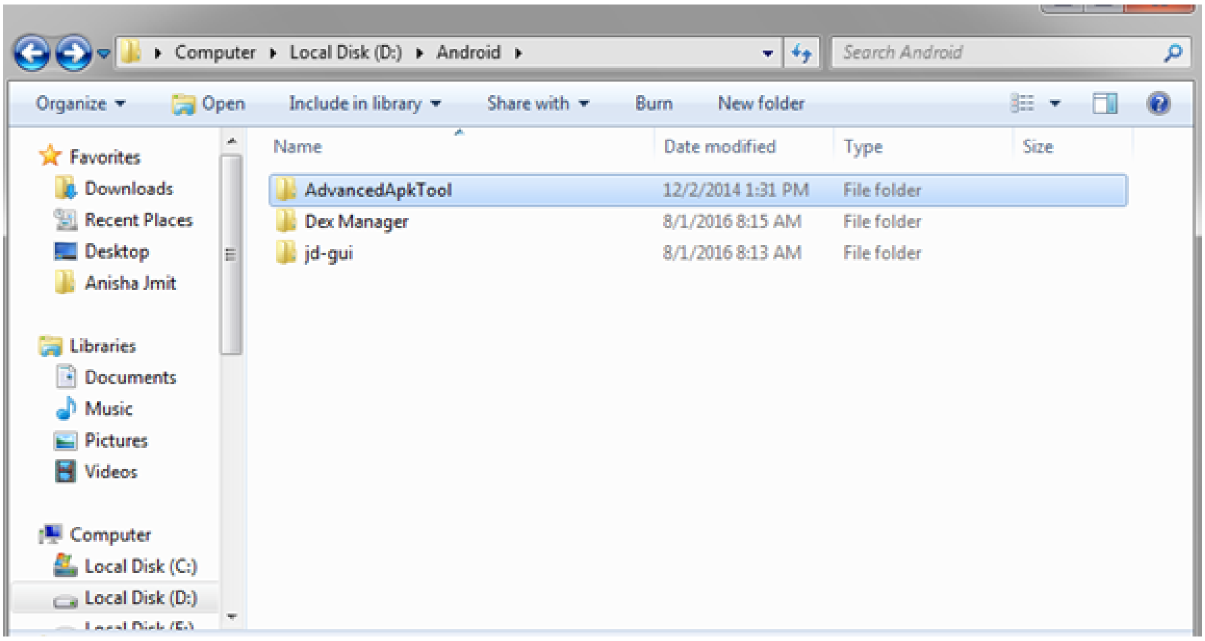
3.2 Proposed Work

Reverse Engineering is a method of analyzing an existing code or piece of software in order to scrutinize the software for any vulnerability or any errors. Reverse engineering is the ability to generate the source code from an executable. This technique is used to scrutinize the functioning of a program or to evade safety mechanisms, etc. Reverse engineering can therefore be stated as a method or process of altering a program in order to make it behave in a manner that the reverse engineer requests. Objectives are:

1. Analysing the possible functionalities of an application using Reverse engineering process
2. To develop two methods to analyse the apk code. The first method involves the utilization of APKTOOL and an editor such as Notepad++. The second method is performed using tools Dex2Jar and JD-GUI.

Setting permissions of an app earlier than they are downloaded. This will results in not completely disabling the app, but will allow users safeguard their privacy and keeps apps from accessing any more user data.

4. RESULTS AND ANALYSIS

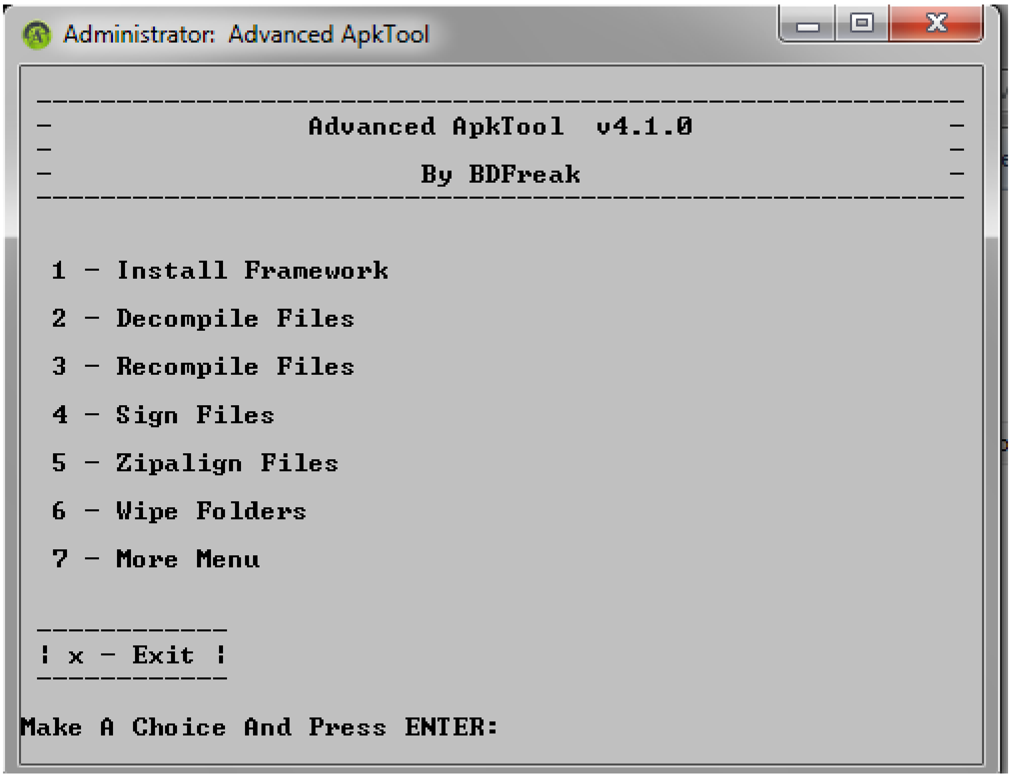


Name	Date modified	Type	Size
AdvancedApkTool	12/2/2014 1:31 PM	File folder	
Dex Manager	8/1/2016 8:15 AM	File folder	
jd-gui	8/1/2016 8:13 AM	File folder	

Figure 4.1 Tools used

Figure 4.1 shows the folder containing AdvancedApkTool, Dex Manager and Jd-gui

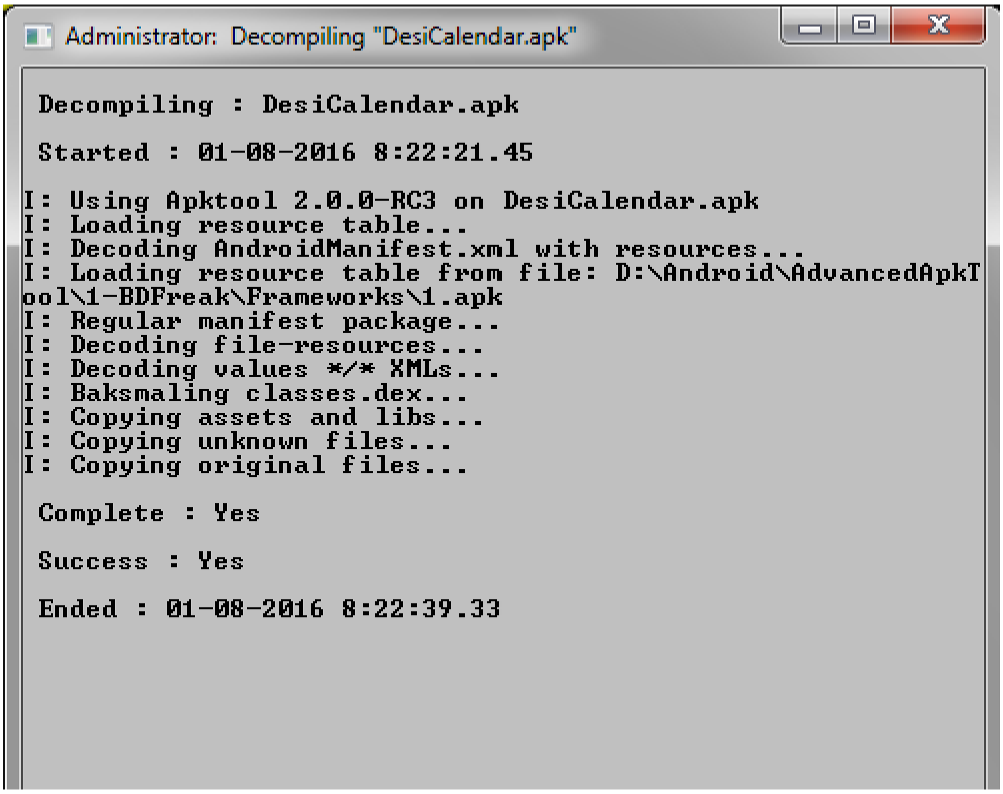
1. ApkTool



```
-----------------------------------------------------------
-                 Advanced ApkTool  v4.1.0                -
-                                                         -
-                      By BDFreak                         -
-----------------------------------------------------------


 1 - Install Framework

 2 - Decompile Files

 3 - Recompile Files

 4 - Sign Files

 5 - Zipalign Files

 6 - Wipe Folders

 7 - More Menu


------------
| x - Exit |
------------

Make A Choice And Press ENTER:
```

Figure 4.2 ApkTool options

Figure 4.2 shows various options available with Advanced ApkTool.



Administrator: Decompiling "DesiCalendar.apk"

```
 Decompiling : DesiCalendar.apk

 Started : 01-08-2016 8:22:21.45

I: Using Apktool 2.0.0-RC3 on DesiCalendar.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: D:\Android\AdvancedApkT
ool\1-BDFreak\Frameworks\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...

 Complete : Yes

 Success : Yes

 Ended : 01-08-2016 8:22:39.33
```

Figure 4.3 Decompiling Apk

Figure 4.3 shows the decompiling the apk file, Decoding AndroidManifest.xml file, Baksmaling classes.dex file.

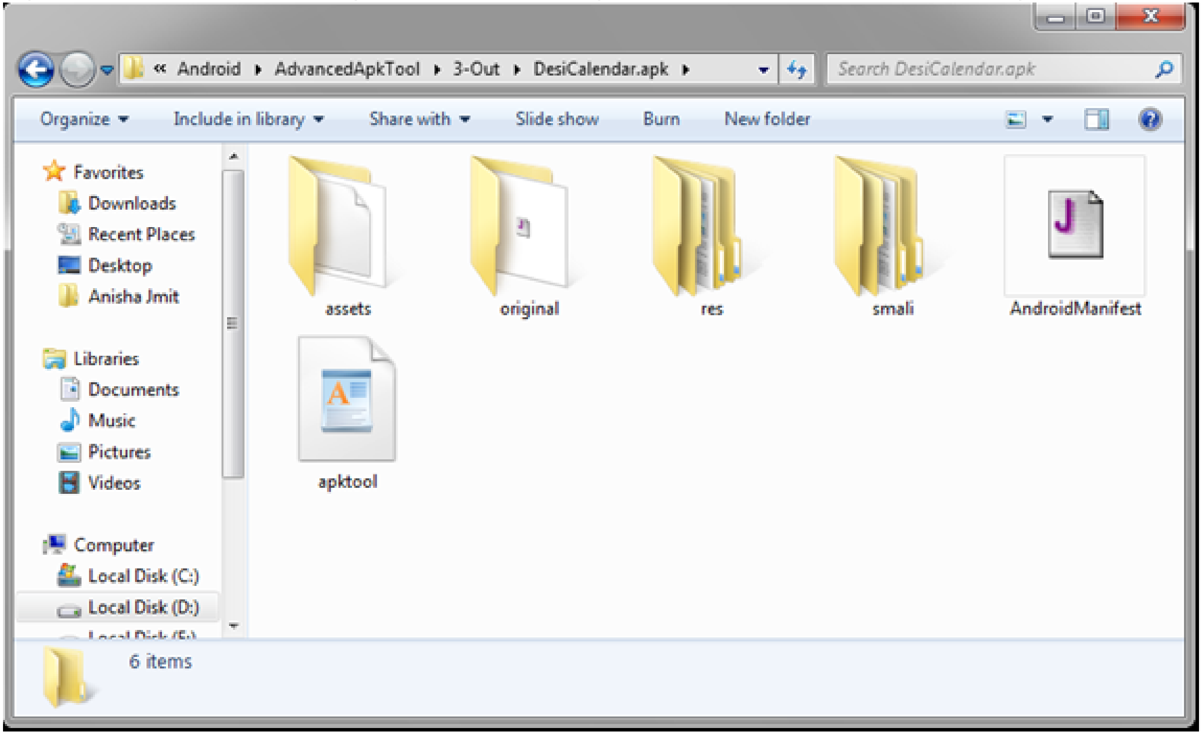
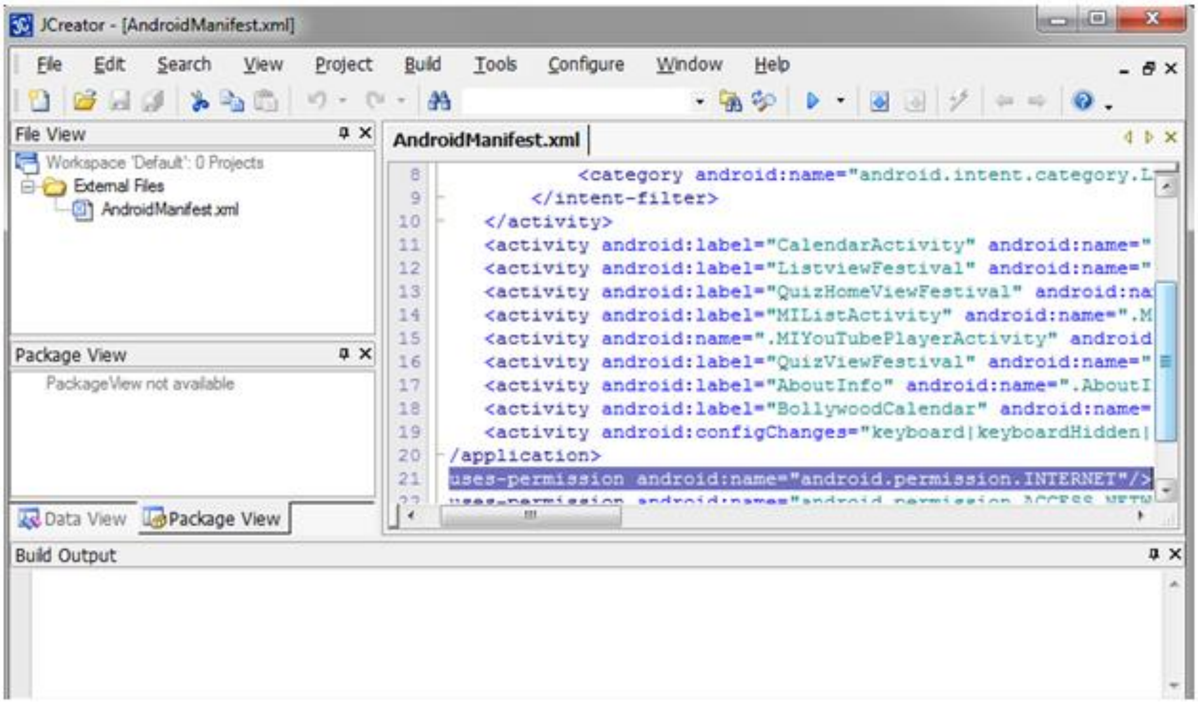


Figure 4.4 Folder containing decompiled apk

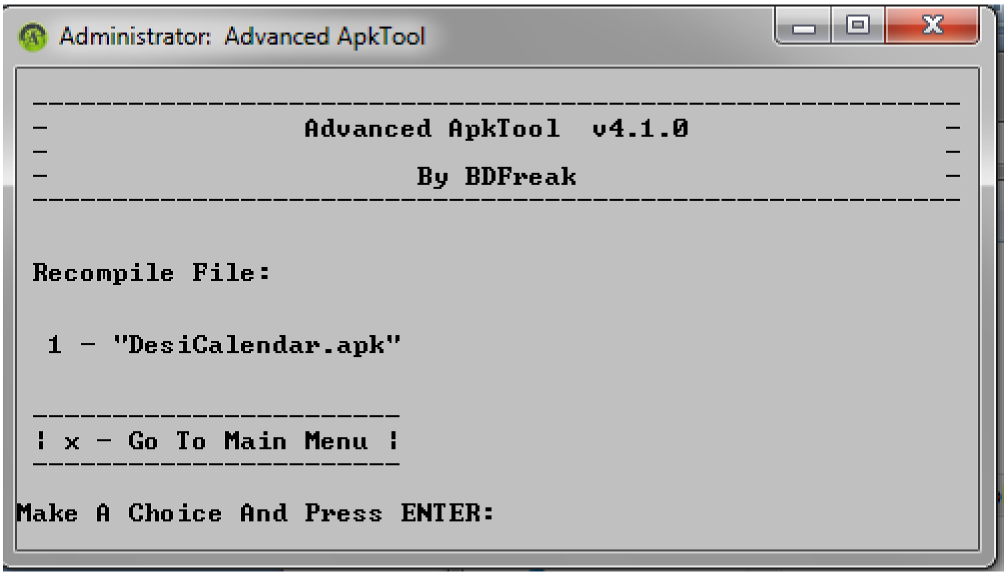
Figure 4.4 shows the folder where decompiled apk is extracted. Now you can edit the AndroidManifest.xml and smali folder.


JCreator - [AndroidManifest.xml]
File Edit Search View Project Build Tools Configure Window Help
File View
Workspace 'Default': 0 Projects
External Files
AndroidManifest.xml
Package View
PackageView not available
Data View
Package View
AndroidManifest.xml
Build Output

```
                <category android:name="android.intent.category.L
            </intent-filter>
        </activity>
        <activity android:label="CalendarActivity" android:name="
        <activity android:label="ListviewFestival" android:name="
        <activity android:label="QuizHomeViewFestival" android:na
        <activity android:label="MIListActivity" android:name=".M
        <activity android:name=".MIYouTubePlayerActivity" android
        <activity android:label="QuizViewFestival" android:name="
        <activity android:label="AboutInfo" android:name=".AboutI
        <activity android:label="BollywoodCalendar" android:name=
        <activity android:configChanges="keyboard|keyboardHidden|
/application>
uses-permission android:name="android.permission.INTERNET"/>
uses-permission android:name="android.permission.ACCESS_NETW
```

Figure 4.5 Revoking Internet permission from apk

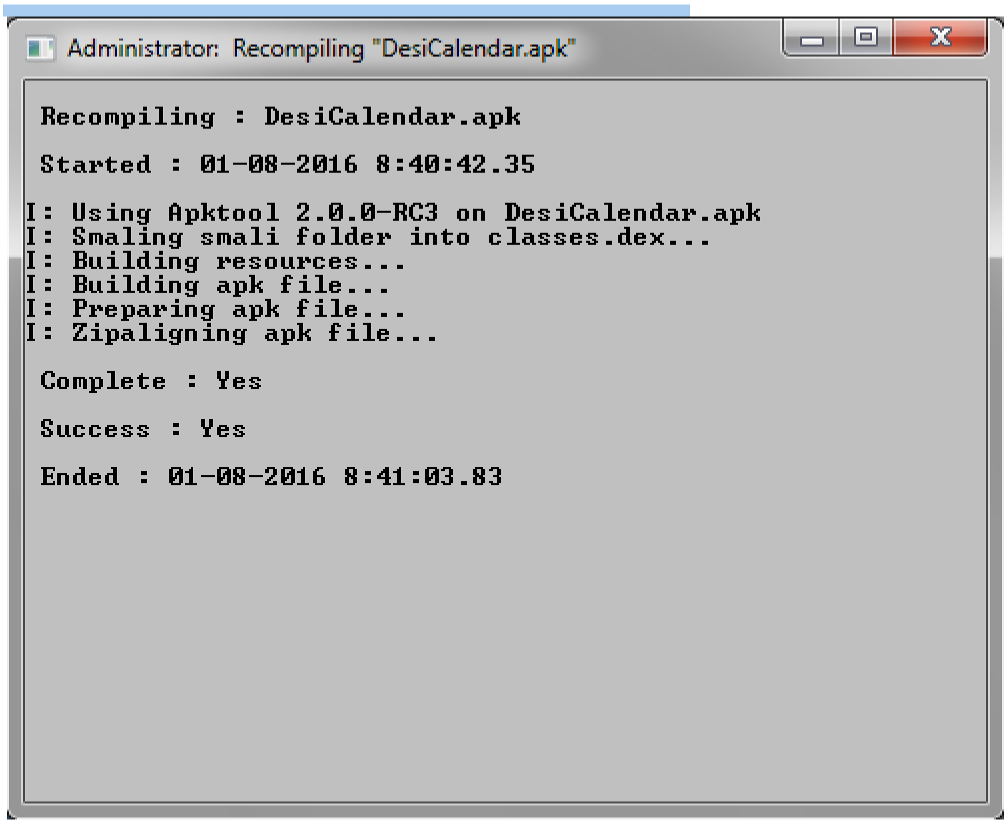
Figure 4.5 shows contents of AndroidManifest.xml opened in Notepad. User can edit this file to revoke a particular permission from apk. The AndroidManifest.xml file contains all the permissions and metadata linked to the security enforcement policy. The tag <Permission> indicates the components that can access it and <Intent-Filters> tag is used to specify the intents that can be resolved.



```
 -----------------------------------------------------------
 -               Advanced ApkTool  v4.1.0                  -
 -                                                         -
 -                      By BDFreak                         -
 -----------------------------------------------------------


 Recompile File:

  1 - "DesiCalendar.apk"


 -----------------------
 ¦ x - Go To Main Menu ¦
 -----------------------

Make A Choice And Press ENTER:
```

Figure 4.6 Recompile apk

Figure 4.6 shows ApkTool recompiling the apk file.



```
 Recompiling : DesiCalendar.apk

 Started : 01-08-2016 8:40:42.35

I: Using Apktool 2.0.0-RC3 on DesiCalendar.apk
I: Smaling smali folder into classes.dex...
I: Building resources...
I: Building apk file...
I: Preparing apk file...
I: Zipaligning apk file...

 Complete : Yes

 Success : Yes

 Ended : 01-08-2016 8:41:03.83
```

Figure 4.7 Recompiling Apk

Figure 4.7 shows recompiling the apk file , building classes.dex from smali folder, building apk file, Zipaligining apk file.

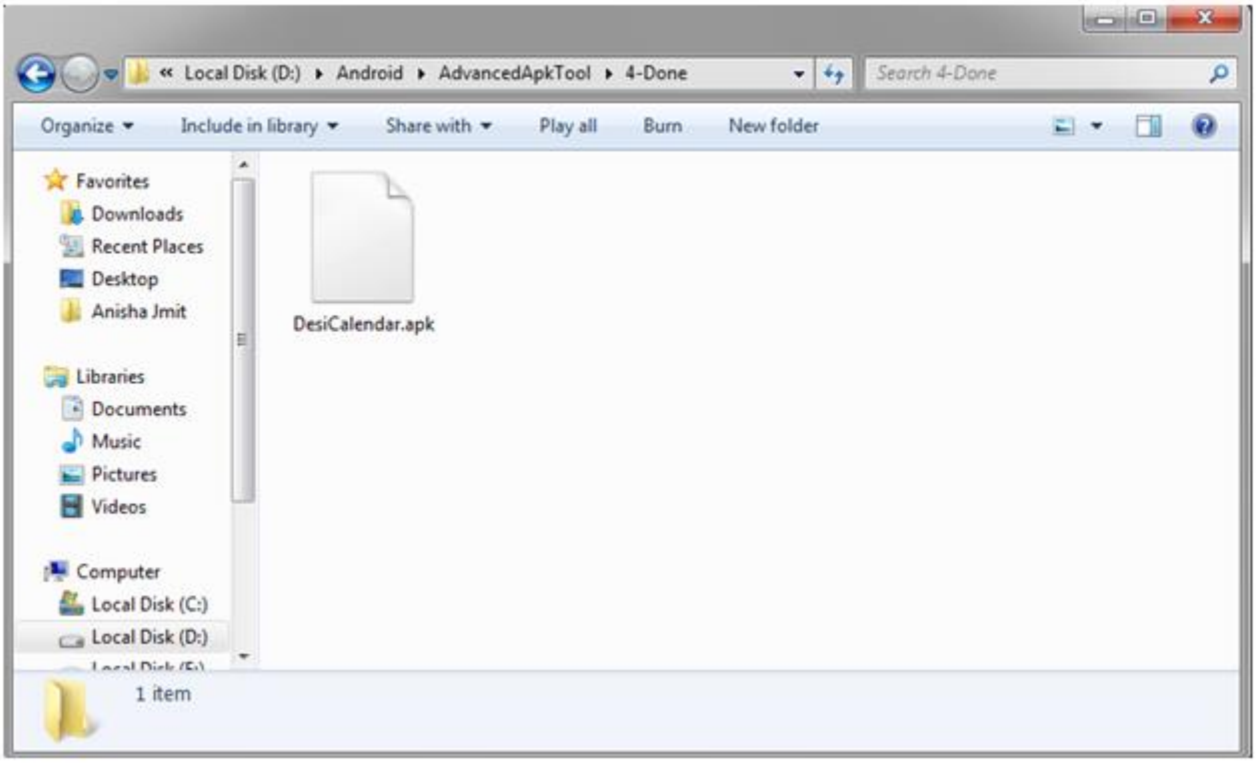
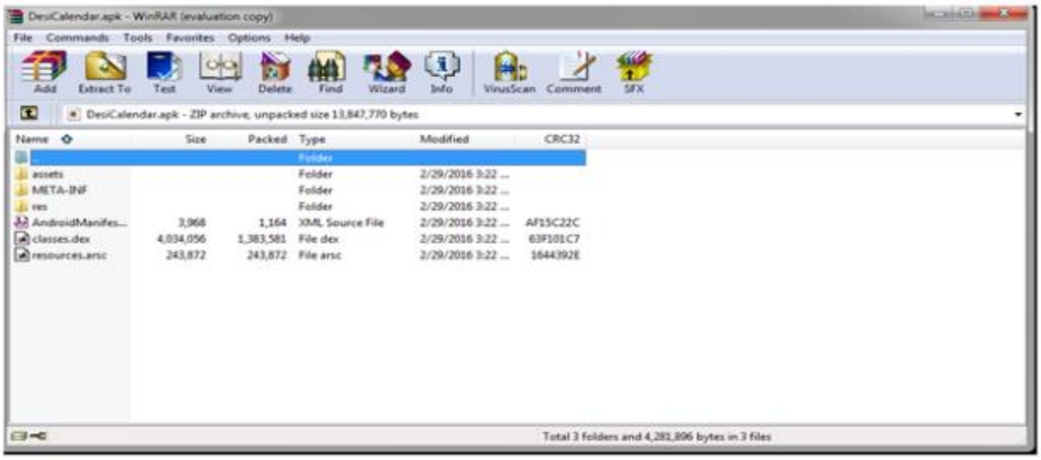


Figure 4.8 Folder containing recompiled apk

Figure 4.8 shows the folder containing the new apk file which is now with restricted permissions and can be installed on Android phone.

2. **Dex Manager**



DesiCalendar.apk - WinRAR (evaluation copy)

DesiCalendar.apk - ZIP archive, unpacked size 13,847,770 bytes

Name	Size	Packed	Type	Modified	CRC32
..			Folder		
assets			Folder	2/29/2016 3:22 ...	
META-INF			Folder	2/29/2016 3:22 ...	
res			Folder	2/29/2016 3:22 ...	
AndroidManifes...	3,968	1,164	XML Source File	2/29/2016 3:22 ...	AF15C22C
classes.dex	4,034,056	1,383,581	File dex	2/29/2016 3:22 ...	63F101C7
resources.arsc	243,872	243,872	File arsc	2/29/2016 3:22 ...	1644392E

Total 3 folders and 4,281,896 bytes in 3 files

Figure 4.9 Winrar to open apk

Figure 4.9 shows the contents of apk file in winrar. Here classes.dex file is required.

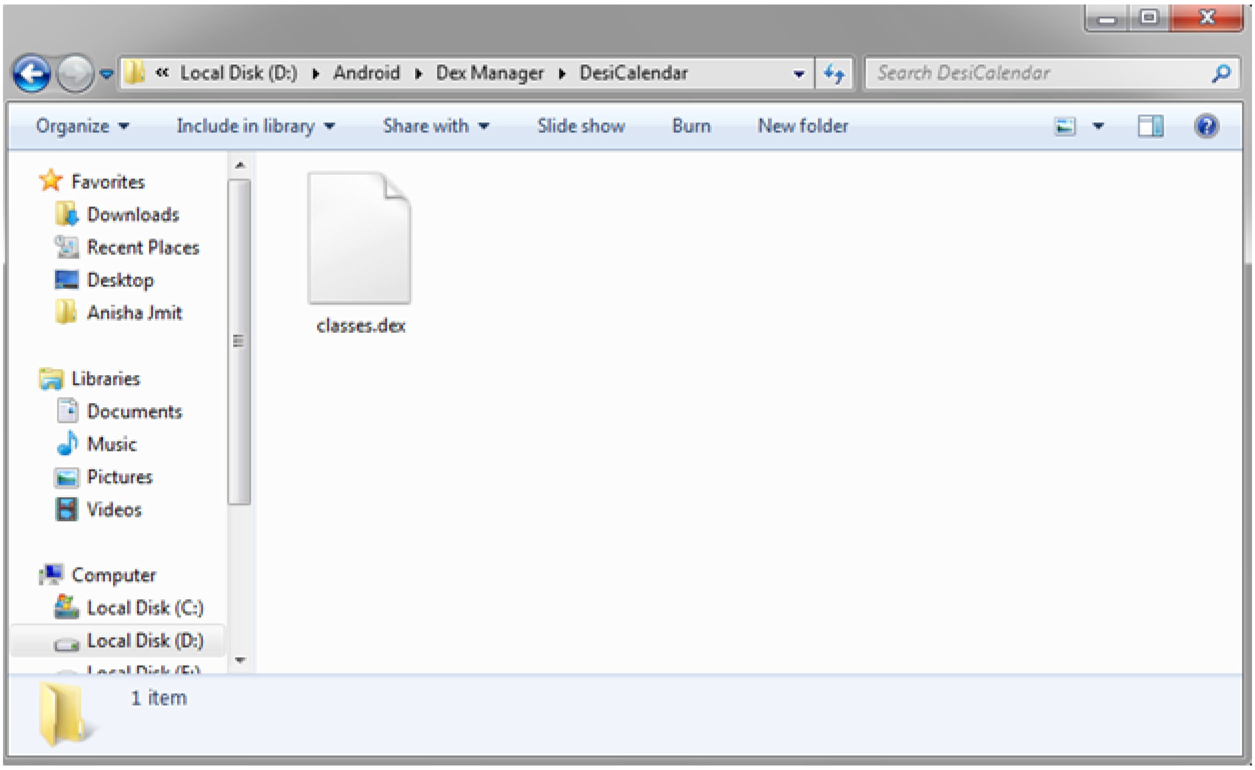
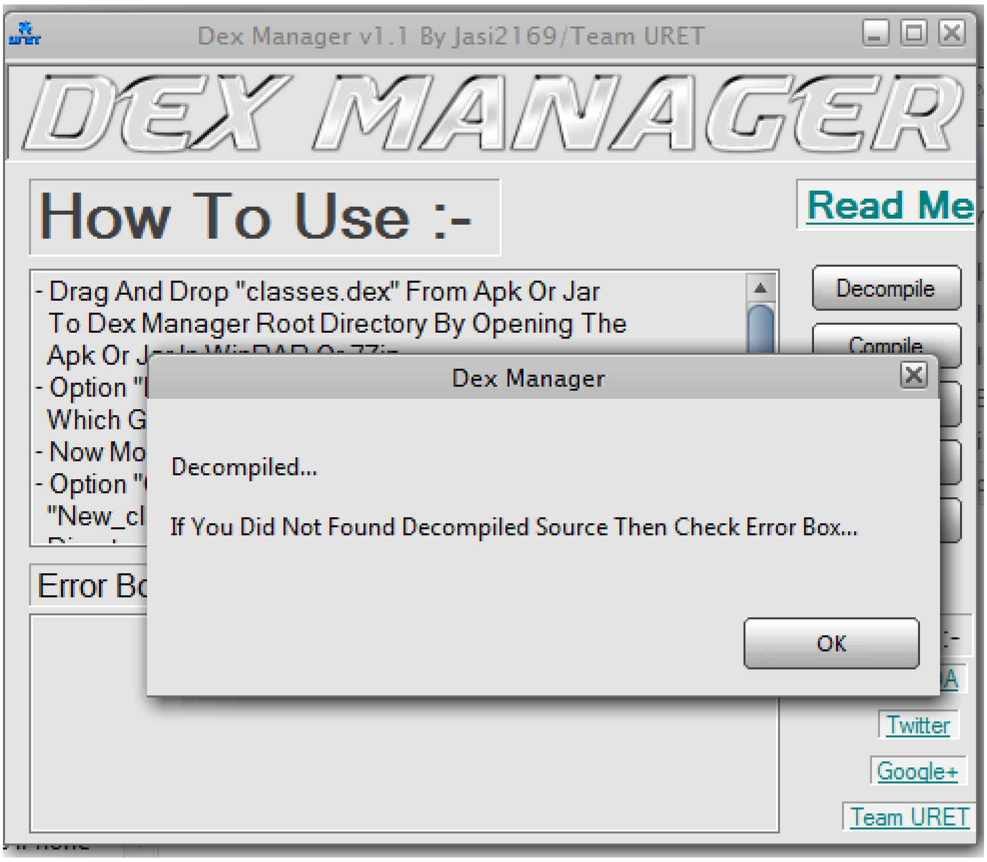


Figure 4.10 Extracting the classes.dex file from apk

Figure 4.10 show the folder containing classes.dex file. Now we will extract source code from the dex file.



- Drag And Drop "classes.dex" From Apk Or Jar To Dex Manager Root Directory By Opening The Apk Or J
- Option "
Which G
- Now Mo
- Option "
"New_cl

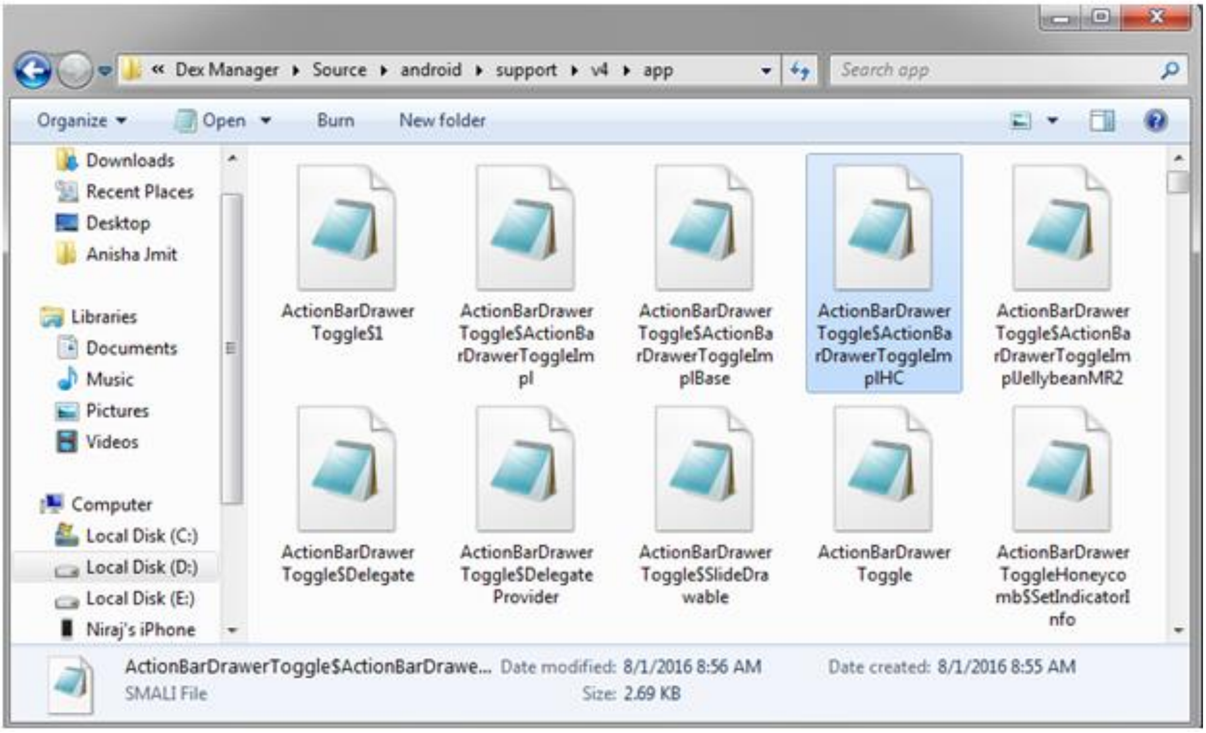
Dex Manager

Decompiled...

If You Did Not Found Decompiled Source Then Check Error Box...

OK

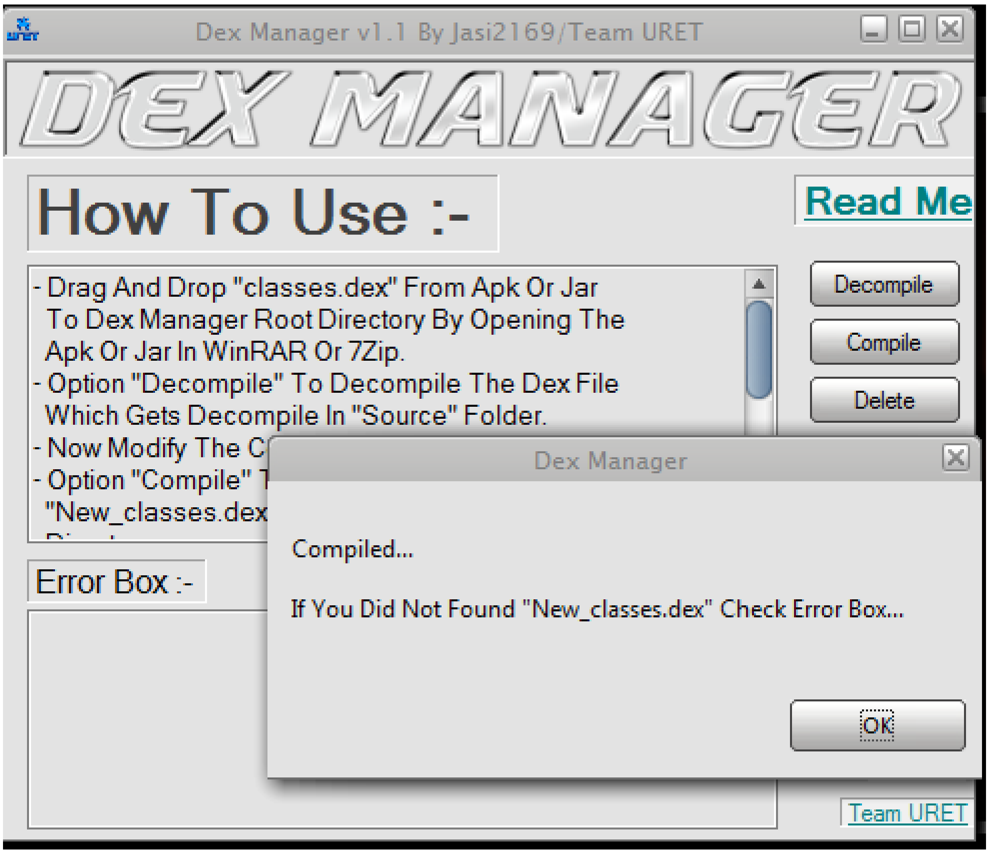
Figure 4.11 Decompiled source code


« Dex Manager ▸ Source ▸ android ▸ support ▸ v4 ▸ app
Search app
Organize ▾
Open ▾
Burn
New folder
Downloads
Recent Places
Desktop
Anisha Jmit
Libraries
Documents
Music
Pictures
Videos
Computer
Local Disk (C:)
Local Disk (D:)
Local Disk (E:)
Niraj's iPhone
ActionBarDrawerToggle$1
ActionBarDrawerToggle$ActionBarDrawerToggleImpl
ActionBarDrawerToggle$ActionBarDrawerToggleImplBase
ActionBarDrawerToggle$ActionBarDrawerToggleImplHC
ActionBarDrawerToggle$ActionBarDrawerToggleImplJellybeanMR2
ActionBarDrawerToggle$Delegate
ActionBarDrawerToggle$DelegateProvider
ActionBarDrawerToggle$SlideDrawable
ActionBarDrawerToggle
ActionBarDrawerToggleHoneycomb$SetIndicatorInfo

ActionBarDrawerToggle$ActionBarDrawe... Date modified: 8/1/2016 8:56 AM Date created: 8/1/2016 8:55 AM
SMALI File Size: 2.69 KB

Figure 4.12 Folder containing source code of apk

Figure 4.12 shows the folder containing source code extracted by Dex Manager. Make changes in source code and save all files in the same folder.



How To Use :-

- Drag And Drop "classes.dex" From Apk Or Jar To Dex Manager Root Directory By Opening The Apk Or Jar In WinRAR Or 7Zip.
- Option "Decompile" To Decompile The Dex File Which Gets Decompile In "Source" Folder.
- Now Modify The C
- Option "Compile" T "New_classes.dex

Error Box :-

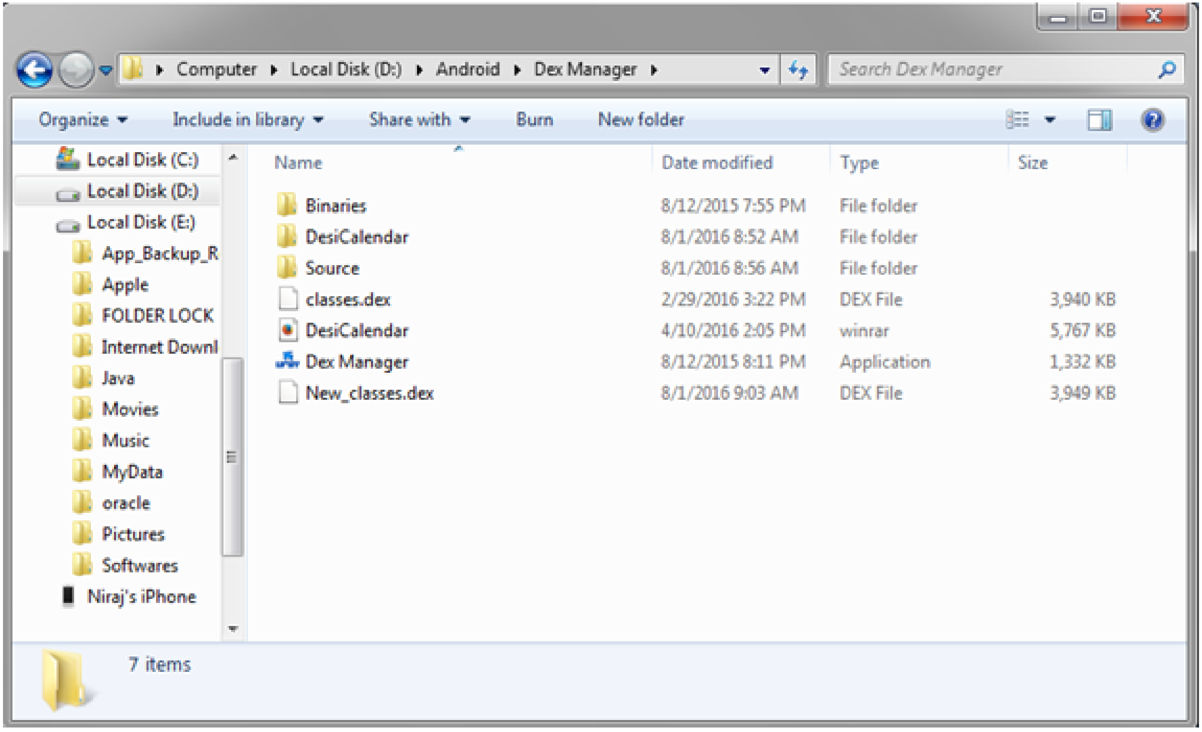
Dex Manager

Compiled...

If You Did Not Found "New_classes.dex" Check Error Box...

Figure 4.13 Compile apk

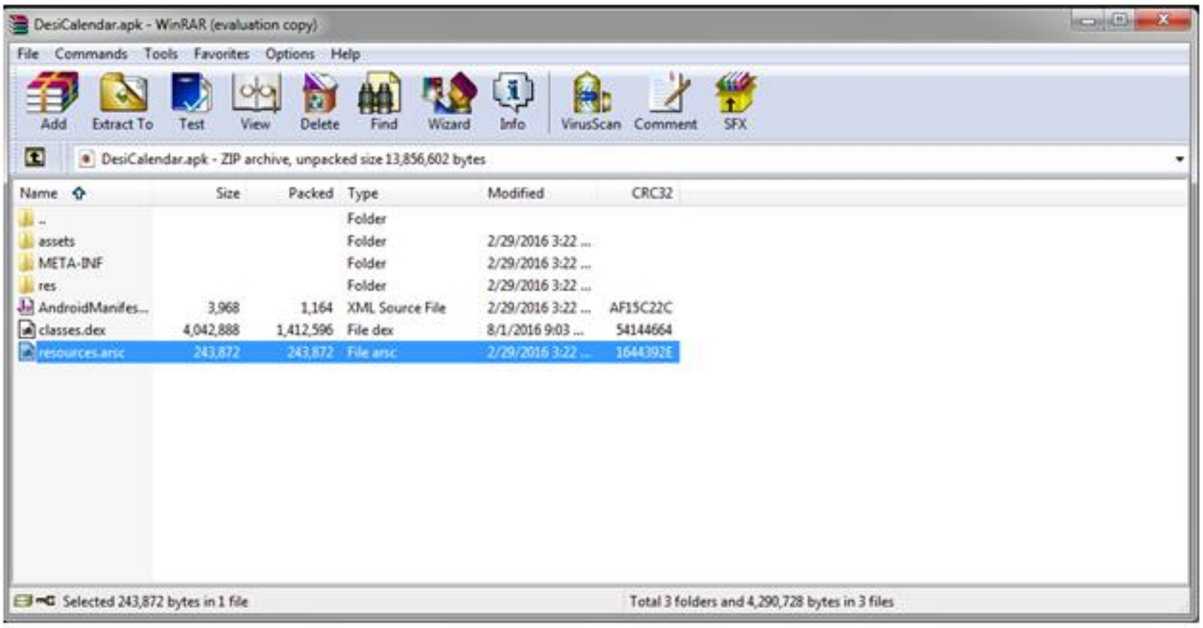
Figure 4.13 shows the compilation of source code to generate the New_classes.dex file.



Name	Date modified	Type	Size
Binaries	8/12/2015 7:55 PM	File folder	
DesiCalendar	8/1/2016 8:52 AM	File folder	
Source	8/1/2016 8:56 AM	File folder	
classes.dex	2/29/2016 3:22 PM	DEX File	3,940 KB
DesiCalendar	4/10/2016 2:05 PM	winrar	5,767 KB
Dex Manager	8/12/2015 8:11 PM	Application	1,332 KB
New_classes.dex	8/1/2016 9:03 AM	DEX File	3,949 KB

Figure 4.14 New_classes.dex generated after compile

Figure 4.14 shows classes.dex and New_classes.dex file. Remove classes.dex and rename New_classes.dex as classes.dex. Copy this classes.dex to apk.


DesiCalendar.apk - WinRAR (evaluation copy)
File Commands Tools Favorites Options Help
Add Extract To Test View Delete Find Wizard Info VirusScan Comment SFX

DesiCalendar.apk - ZIP archive, unpacked size 13,856,602 bytes

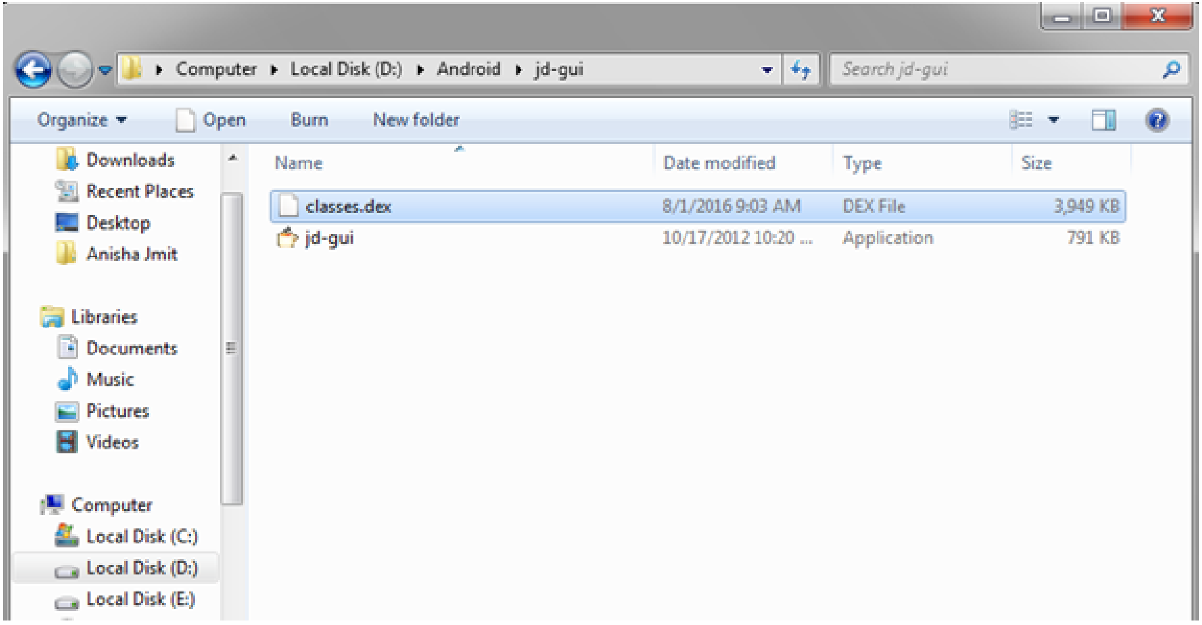
Name	Size	Packed	Type	Modified	CRC32
..			Folder		
assets			Folder	2/29/2016 3:22 ...	
META-INF			Folder	2/29/2016 3:22 ...	
res			Folder	2/29/2016 3:22 ...	
AndroidManifes...	3,968	1,164	XML Source File	2/29/2016 3:22 ...	AF15C22C
classes.dex	4,042,888	1,412,596	File dex	8/1/2016 9:03 ...	54144664
resources.arsc	243,872	243,872	File arsc	2/29/2016 3:22 ...	1644392E

Selected 243,872 bytes in 1 file Total 3 folders and 4,290,728 bytes in 3 files

Figure 4.15 Modified apk

Figure 4.15 shows the Modified apk to which now new classes.dex file has been added.

3 Jd-gui



Name	Date modified	Type	Size
classes.dex	8/1/2016 9:03 AM	DEX File	3,949 KB
jd-gui	10/17/2012 10:20 ...	Application	791 KB

Figure 4.16 jd-gui folder

Figure 4.16 shows the folder containing jd-gui tool and classes.dex file from which source code is to be extracted.

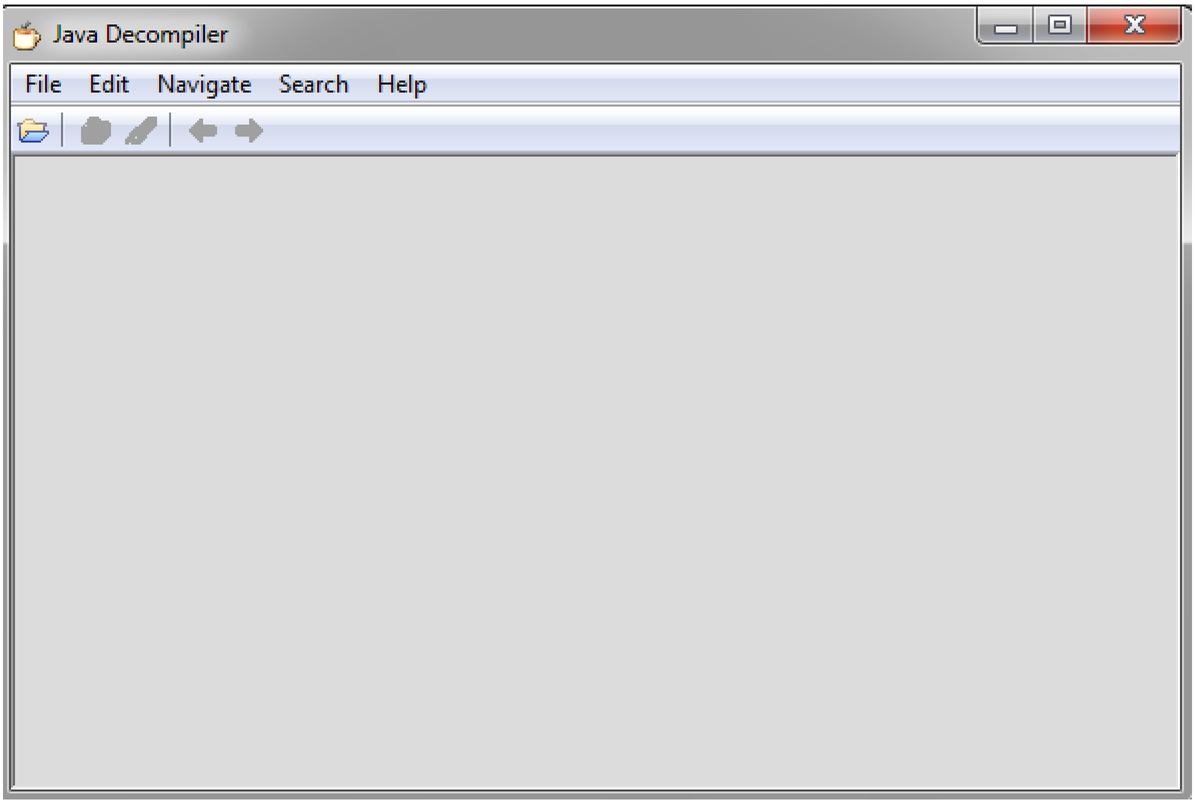
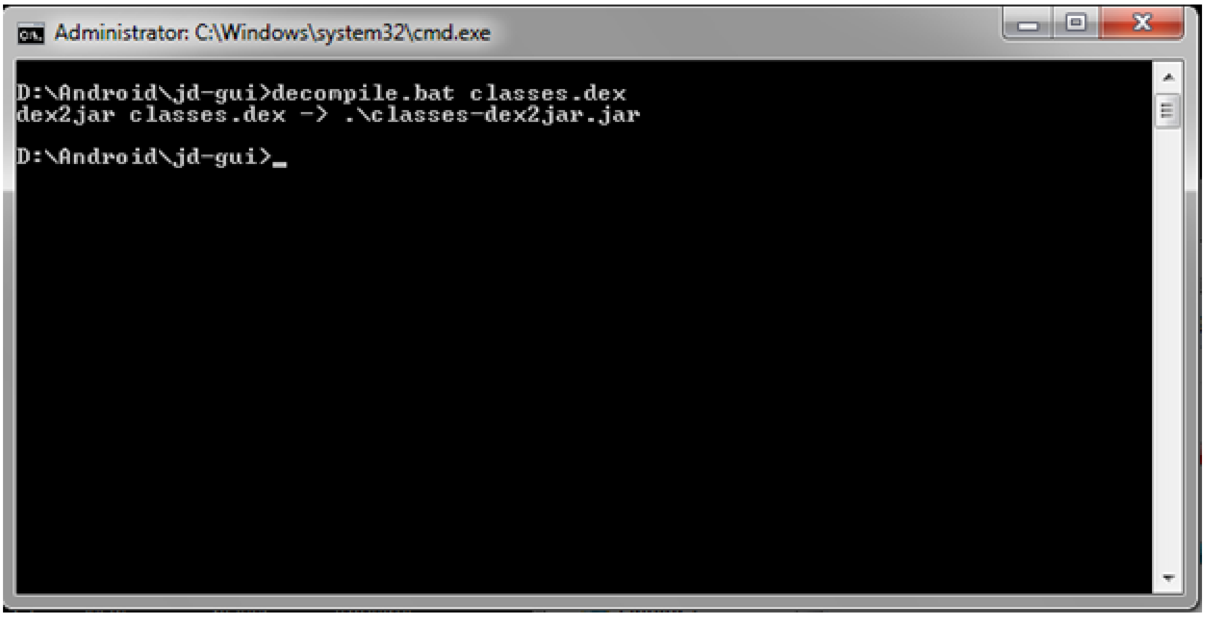


Figure 4.17 Java Decompiler

Figure 4.17 shows various options of Java Decompiler.



```
D:\Android\jd-gui>decompile.bat classes.dex
dex2jar classes.dex -> .\classes-dex2jar.jar

D:\Android\jd-gui>_
```

Figure 4.18 Convert classes.dex to jar

Figure 4.18 shows the conversion of classes.dex file to jar. This jar file can currently be opened in jd-gui.

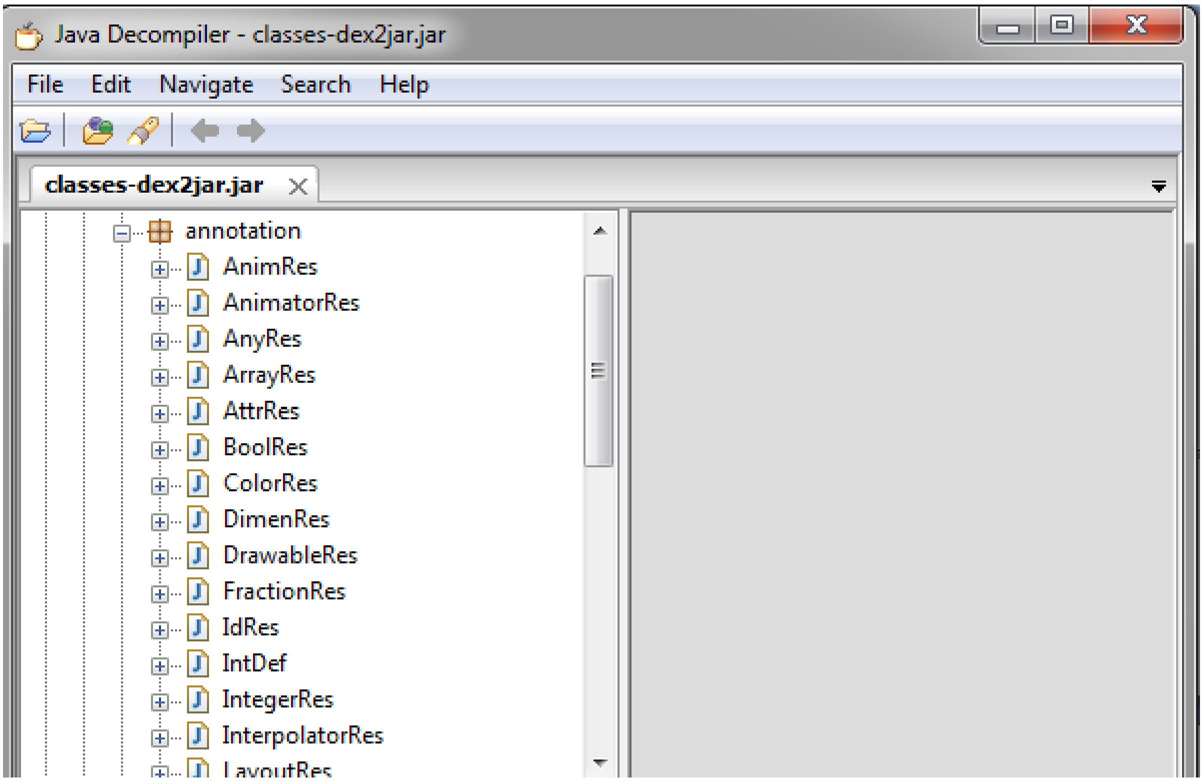


Figure 4.19 Source code of apk

Figure 4.19 shows the java source code of apk which can now be edited.

5. Conclusion

From the analysis made in above, we can confirm that special attention needs to be provided for the permissions that an application requests access to. The user must decide if these permissions are really required by the application or not. Just as there are hackers/attackers releasing malwares for PCs, there are attackers who are now targeting smart phones. The main reason for this is that mobile security is still in its initial stages and lack of user alertness regarding how these devices can be compromised if they are not careful enough. In this project the tools we use for reverse engineering are, ApkTool, DexManager, and Dex2Jar, JD-GUI. The several malwares that exist in the Android platform are a ground of concern for both the users as well as Google. Existing work can be extended to further analyze the malwares, their effects and how they can be eradicated in order to provide malware free applications for users.

References

[1]. Quang Do, Ben Martini, Kim-Kwang, Raymond ChooEnhancing, “User Privacy on Android Mobile Devices via Permissions Removald”, 2014 47th Hawaii International Conference on System Science.

[2]. Andr´eEgners, Ulrike Meyer, Bj¨ornMarschollek, ”Messing with Android’s Permission Model”, © 2012 IEEE

[3]. ChenkaiGuo, Jing Xu, Lei Liu and SihanXu,”MalDetector-Using Permission Combinations to Evaluate Malicious Features of Android App”, ©2015 IEEE.

[4]. GarimaBajwa, Mohamed Fazeen, Ram Dantu and Sonal Tanpure,”Unintentional Bugs to Vulnerability Mapping in Android Applications”, ©2015 IEEE.

[5]. Mario Frank, Ben Dong, Adrienne Porter Felt, Dawn Song, ”Mining Permission Request Patterns from Android and Facebook Applications (extended author version)”, arXiv:1210.2429v1 [cs.CR] 8Oct 2012.

[6]. ParvezFaruki, Vijay Laxmi, Vijay Ganmoor, M.S. Gaur, Ammar Bharmal, ”DroidOLytics : Robust Feature Signature for Repackaged Android apps on Official and Third party android markets”, © 2013 IEEE

[7]. Wook Shin, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka, ”A Formal Model to Analyze the Permission Authorization and Enforcement in the Android Framework”, © 2010 IEEE

[8]. Hamid Bagheri, Alireza Sadeghi, Joshua Garcia and Sam Malek,”COVERT: Compositional Analysis of Android Inter-App Permission Leakage”, 2015. IEEE Transactions on Software Engineering.